JQUERY WEB-SOVELLUKSEN APUVÄLINEENÄ

HAMK
HÄMEEN AMMATTIKORKEAKOULU

Ammattikorkeakoulun opinnäytetyö

Tietotekniikka

Riihimäki, kevät 2015

Veli-Matti Pyykkönen

HAMK
HÄMEEN AMMATTIKORKEAKOULU

RIIHIMÄKI
Tietotekniikka
Ohjelmistotekniikka

Tekijä Veli-Matti Pyykkönen **Vuosi** 2015

Työn nimi jQuery web-sovelluksen apuvälineenä

TIIVISTELMÄ

Opinnäyte perustuu työharjoittelussa asiakasyritykselle tehtyyn sovellukseen, jonka yhteydessä tutustuttiin itsenäisesti jQueryyn. Työssä opittiin ja sovellettiin jQueryn käyttämistä yhdessä JavaScriptin kanssa.

Työn tavoitteena oli paneutua syvemmin jQueryn tarjoamiin toimintoihin ja tutkia sen käyttämistä. Työssä sovellettiin koulutuksessa opittuja taitoja. Pääasiassa täytyi osata JavaScriptin ja jQueryn käyttöä, mutta myös HTML:n ja CSS:n käyttämisen piti olla tuttua. Itse jQueryn käyttäminen opittiin itse, mutta syntaksin ymmärtämisen jälkeen sen käyttö oli tutumpaa ja noudatti muista ohjelmointikielistä tuttuja konventioita.

Kaikessa jQueryn komentoihin liittyvissä asioissa käytettiin jQueryn virallisia dokumentointeja. Muihin lähteisiin, esimerkiksi artikkeleihin, turvauduttiin kun tarvittiin tietoja, jotka eivät liittyneet suoraan jQueryn toimintaan. Koodipätkiä tuli kokeiltua myös itse ja sijoitettua niitä omaan työhön kehitysmielessä tapauksissa, joissa uusista opituista asioista koki tulevan lisäarvoa omalle työlle.

jQueryn toiminnasta avattiin valitsimien käyttö, CSS ominaisuuksien muuttaminen ja animaatiot, käytiin läpi hiiritapahtumat sekä DOM-elementtien lisääminen, poistaminen, korvaaminen ja kloonaaminen.

jQuerystä paljastui tehokas ja syystäkin suosittu työkalu, joka auttaa kehittäjiä työssään. Suorituskyvyssä se ei pärjää puhtaalle JavaScriptille, mikä pitää ottaa huomioon, jos haetaan parasta mahdollista suorituskykyä, mutta työkalut tulisi aina valita kunkin työtehtävän mukaan.

Avainsanat jQuery, Dokumenttiobjektimalli (DOM), JavaScript

Sivut 38 s.

HAMK
UNIVERSITY OF APPLIED SCIENCES

RIIHIMÄKI
Degree programme in Information Technology
Programming Technology

Author Veli-Matti Pyykkönen **Year** 2015

Subject of Bachelor's thesis jQuery as a tool in web application

ABSTRACT

This thesis is based on a project that was made for a client company during the internship of the author. During the project we started to explore the use of jQuery. In this project we learned to use and apply jQuery together with JavaScript.

The objective of this thesis project was to gain a deeper knowledge of jQuery and to learn about its uses and the possibilities it offers. During the project we applied our skills learned during our education at HAMK. For this project the use of jQuery was learned independently. After having learnt its syntax it started to feel more familiar as it followed the logic of other programming languages.

In everything that had to do with the commands of jQuery, the official documentation was used. Other sources such as articles were used when the information needed did not involve the functions of jQuery. Some of the code shown here was also integrated within the original application when appropriate and if it added something of value to the program.

In order to use jQuery one has to have knowledge in JavaScript, HTML and CSS. Of the main functions of jQuery the use of selectors, changing the CSS values, animations, mouse events, as well as adding, removing, replacing and cloning of DOM elements are covered here.

jQuery turned out to be a very powerful and popular tool that offers much for the developers using it. It does not rise to the level of JavaScript in its performance, however, and that is something one must be aware of when choosing the appropriate tools for the task.

Keywords jQuery, Document Object Model (DOM), JavaScript

Pages 38 p.

SISÄLLYS

1 JOHDANTO 1

2 ALKUPERÄINEN OHJELMA 1

3 TARVITTAVAT TIEDOT JA TEKNIIKAT 3

3.1 Document Object Model 3
3.2 JavaScript 3
3.3 HTML 4
3.4 CSS 4

4 JQUERY JA SEN KÄYTTÖ 4

4.1 jQueryn synty 4
4.2 jQueryn käyttöönotto 5
4.3 Miksi jQuerya kannattaa käyttää 6
4.4 Esimerkkejä jQueryn käytöstä 6
4.4.1 Nimettömien funktioiden välttäminen 8
4.4.2 Optimointikikkoja 9
4.4.3 $(Document).ready()-metodi 10
4.5 Valitsimet 10
4.5.1 Perusvalitsimet 11
4.5.2 Attribuuttivalitsimet 12
4.5.3 Sisältövalitsimet 13
4.5.4 Perusvalitsimien suodattimia 13
4.5.5 Piilotetut ja näkyvät elementit 14
4.6 CSS-ominaisuudet 15
4.7 jQueryn animaatiot 16
4.7.1 Haihdutus 17
4.7.2 Liu’utus 18
4.7.3 Perustehoste-komennot 19
4.8 Hiiritapahtumat 20
4.9 Elementtien lisääminen 22
4.9.1 Ulkopuolelle 22
4.9.2 Sisäpuolelle 23
4.9.3 Ympäri 24
4.9.4 DOM-elementtien poistaminen 25
4.9.5 DOM-elementtien korvaaminen 26
4.9.6 Elementtien kloonaus 27
4.10 jQueryn lisäosat 28
4.10.1 jQuery UI 28
4.10.2 jQuery Mobile 29

5 JQUERYN HYÖDYT JA HAITAT 30

6 YHTEENVETO 32

LÄHTEET 33

1 JOHDANTO

Osallistuin 2014 keväällä projektiin, jossa tehtiin asiakasyritykselle mobiilisovellus, jonka vaatimuksena oli, että se tehtiin web-tekniikoin. Nämä tekniikat ovat perinteisesti HTML5, CSS ja JavaScript. Työtä tehtäessä ja etsittäessä ohjeita JavaScript-ongelmiin saatiin neuvoja, joissa puhuttiin, miten asia tehtäisiin JavaScript-kirjastolla nimeltä jQuery, josta ei ollut minkäänlaista kokemusta.

Koska kyseessä oli pelkkä JavaScript-kirjasto, se lasketaan myös web-tekniikoihin. Työryhmässä päätettiin tutustua asiaan tarkemmin ja hyödyntää sitä. Kävi selväksi, että jotkin asiat tapahtuivat jQueryllä huomattavasti helpommin kuin JavaScriptillä. Esimerkiksi koodi on lyhyempää ja komennot selkeitä.

jQuery on vakiinnuttanut paikkansa yhtenä suosituimmista JavaScript-kirjastoista. Vuoden 2015 alussa jQuerya käytti noin 61,5 % kaikista web-sivuista. (W3Techs 2015). On hyvin todennäköistä, että tämän päivän web-kehittäjä tulee törmäämään tähän lisäosaan ja tämän lisäosan tuntemusta tullaan tarvitsemaan.

Projektin aikana luotiin vain pieni katsaus jQueryn ominaisuuksiin, eikä pienen käytön seurauksena voinut sanoa, että hallitsisi tämän kirjaston käytön ja sen, mitä tarkoitusta varten se on luotu. Kun kaiveli lisää, selvisi, että kyseessä on varsin suosittu kirjasto, jolla on hurjasti tukea takanaan suurilta tietotekniikan jättiläisiltä. Voisi sanoa, että jQuery on elänyt sen vaiheen ohi, jossa se olisi nauttinut suosiota lyhyen aikaa ja sen jälkeen kuollut pois, kuten käy monille tietotekniikan uusille trendeille.

Koin, että jQuery olisi tärkeä tutkimuskohde ja olisi syytä selvittää sen ominaisuudet ja käyttötarkoitus. Tähän kaikkeen ei olisi aika riittänyt, joten näkökulma on rajattu siten, miten jQuerya hyödynnettiin projektin aikana apuvälineenä täydentämään JavaScriptiä. Lisäksi pohdin jQueryn hyötyjä ja haittoja.

2 ALKUPERÄINEN OHJELMA

Asiakasyrityksellä oli käytössään HAMKin kanssa aiemmin tehty tuotekatalogi, mutta se oli tehty Adoben tekniikoilla eikä näin toiminut mobiililaitteissa sellaisenaan. Tavoitteena oli tehdä tästä ohjelmasta versio, jonka toiminta olisi mahdollisimman alustasta riippumaton. Toiveena oli, että alkuperäisen ohjelman puurakenne-käyttöliittymä säilytettäisiin sellaisenaan mikäli mahdollista.

Alkuperäisen ohjelman tarkoitus oli näyttää Excel-taulukon tiedot laitteessa, johon ei saa Exceliä, eli mobiililaitteessa. Ohjelma toteutettiin JavaScriptillä eli laitteesta täytyy löytyä sitä tukeva internetselain. Excel-tiedosto itsessään pitää sisällä yrityksen erinäisten laitteiden tietoja, kuten sarjanumeroita ja tuoteperheiden erilaisia malleja. Excel-tiedoston täytyy olla

myös CSV-muodossa, jolloin se on käytännössä pelkkä tekstitiedosto. Excel-tiedostot tulivat asiakasyritykseltä eikä niihin saanut tehdä muutoksia. Ohjelma on toiminnoiltaan varsin yksinkertainen: se tulostaa taulukon esiin ja siitä pystyy karsimaan tietoja nappuloilla, jolloin haluttujen tuotteiden löytäminen on helpompaa. Ohjelma luki CSV-tiedoston pystyrivejä ja laski, montako erilaista nimeä löytyi ja näin monta nappia tarvittiin. Uusi taulukko tulostettiin ja vain tarvittavat tiedot jätettiin näkyviin, loput piilotettiin. Laskettiin uudelleen, montako erilaista nimeä seuraavalta pystyriviltä löytyi, tehtiin napit, tulostettiin taulukko, jne. Jokainen erilainen nimi tallennettiin taulukkoon, josta luettiin tulevien nappien nimet ja lukumäärä.

Ohjelman käyttöliittymä koostui siis napeista, joiden nimet vastasivat taulukosta luettuja tietoja. Näkyvissä olivat vain karsittavissa olevat tiedot ja nappuloista näki, mitkä tiedot oli luettu seuraavalta pystyriviltä ja täten karsittavissa seuraavaksi. Näin nappuloista koostui puurakenne, jonka juuri oli ensimmäisenä painettu nappula. Napit sijaitsevat taulukon yläpuolella ja uudet nappulat tulostuvat uudelle riville edellisten nappien alapuolelle.

Ohjelmaan toivottiin myös ominaisuutta, josta selviäisi yhdellä silmäyksellä, missä kohtaa puurakennetta mennään. Nappuloiden alapuolelle ilmestyy tekstinä puurakenteen tähänastinen reitti eli painettujen nappien nimet eräänlaisena visuaalisena vihjeenä.

Nappulat myös piti pystyä poistamaan tarvittaessa, jos taulukkoa kelattiin taaksepäin. Taaksepäinkelaus saattoi tapahtua taso tasolta tai useampi taso kerralla, joten nappien poiston logiikan suhteen piti olla tarkkana. Todettiin myös tarve napille, joka palauttaa taulukon alkuperäiseen tilaan, jossa koko taulukko on näkyvissä.

Ohjelmaan lisättiin myös hakutoiminto, jolla löytyi halutut tuotteet, jos tiesi tarkalleen, mitä oli hakemassa. Taulukko päivittyi sitä mukaa kun kirjaimia syötti, eli erilaista ”hae”-nappia ei tarvittu.

Ohjelman ollessa melkein valmis, käytiin vielä läpi ohjelman ulkomuotoon liittyviä seikkoja. Taulukkoon luotiin ”seepraraidat”, jolloin joka toinen rivi väritetään eriävällä värillä. Tässä päädyttiin väreihin, jotka olivat olleet aiemmassa katalogi-versiossa. Palautteena tuli, että vanhan version värit olivat silmälle ystävällisemmät kuin ryhmän ehdotus, jonka inspiraationa oli asiakasyrityksen logon väritys. Nappuloiden kokoa jouduttiin säätämään, jotta niiden paineleminen olisi helpompaa kosketusnäyttöisellä laitteella. Taulukon tekstin fonttikooksi laitettiin 12, sillä se vaikutti hyvältä kompromissilta taulukon kokoon suhteutettuna, jolloin vierittämistä sivusuuntaan ei tulisi liikaa ja toisaalta ei tarvitsisi zoomata kohtuuttoman lähelle, että taulukon tekstiä pystyisi myös lukemaan. Taulukon koosta johtuen alaspäin vierittämistä ei oikein voinut välttää, ja nappuloista tietoja karsimalla se pysyi suhteellisen hyvän kokoisena, ainakin sillä Excel-tiedostolla, joka saatiin asiakasyrityksen edustajalta testausta varten. Muilla tiedostoilla ei päästy testaamaan eikä yritykseltä saatu palautetta, joka viittaisi siihen, että taulukon koon kanssa olisi ollut ongelmia.

3 TARVITTAVAT TIEDOT JA TEKNIIKAT

Kun puhutaan koodikirjastosta, joka rakentuu toisen kielen päälle, on hyvä pitää mielessä millaista tietämystä tarvitaan, jotta kirjastosta saisi kaiken irti. jQueryn tapauksessa pitää tietää tekniikat, joita käytetään tämän päivän web-sivuilla ja -sovelluksissa.

3.1 Document Object Model

Document object model, suomeksi dokumenttiobjektimalli eli lyhyesti DOM, on ohjelmointirajapinta, joka pitää kirjaa HTML-dokumentin sisällä olevista elementeistä. DOM rakentaa jokaisesta HTML-dokumentissa esiintyvästä elementistä solmun (engl. Node) puurakenteeseen, jonka alin taso eli juuritaso on document. Kaikki objektit liitetään juurisolmuun. Solmut kuvaavat elementtien välisiä suhteita ja solmulla voi olla lapsi-, isä- tai sisarsolmuja. Jokainen solmu on oma olionsa eli Node-olio (Korpela 2014, 47, 69). Kaikkiin solmuihin pääsee käsiksi JavaScriptillä ja niiden välisiä suhteita voi myös käyttää hyväksi. Koodillisesti JavaScriptissä document.append(child) lisää määrätyn elementin seuraavaan vapaaseen kohtaan HTML-dokumentissa ja DOM:iin luodaan vastaavaa elementtiä edustava uusi solmu. jQuery hyödyntää tätä puurakennetta käydessään läpi dokumentin elementtejä.

3.2 JavaScript

Koska jQuery on JavaScript-kirjasto, on tarpeellista käydä läpi lyhyesti, mikä on JavaScript.

JavaScript on ohjelmointikieli, joka toimii vain selaimessa. Tästä syystä JavaScriptiä kutsutaan asiakaspuolen kieleksi. Se ei vaadi keskustelua serverin kanssa kuten palvelinpuolen kielet ja tästä syystä sillä pystyy antamaan käyttäjälle suoraa palautetta ilman turhia lataustaukoja, kun taas palvelinpuolen ohjelma joutuu keskustelemaan serverin kanssa ja latailemaan tietoja.

JavaScript mahdollistaa HTML-sivujen toiminnan luomisen ja saa sivut toimimaan älykkäästi ja vuorovaikutuksessa käyttäjän kanssa. JavaScript tarjoaa myös työkalut HTML-sivun ulkonäön muokkaamiseen yhdessä CSS:n kanssa.

JavaScriptin avulla saa myös perinteisen web-sivun näyttämään ja toimimaan kuten tavallinen työpöytäsovellus, vaikka se toimiikin ainoastaan selaimessa. JavaScript tarvitsee vain web-selaimen toimiakseen, mikä on ideaalista tehtäessä samankaltaista työtä, jossa käyttöympäristön oli oltava mahdollisimman laiteriippumaton.

3.3 HTML

HTML eli HyperText Markup Language on skriptauskieli, jolla web-sivut kirjoitetaan. Sivulle määritellään elementtejä erilaisin tagein ja nämä määrittelevät sivun tekstisisällön. Tagien ulkonäköä taas säädellään CSS:llä. Selain kääntää tämän skriptin lennosta ja lopputulos näyttää perinteiseltä internetsivulta. Työssä käytettiin HTML5-version mahdollistamaa File API:n alta löytyvää Filereader-rajapintaa, joka mahdollistaa käyttäjän valitseman tekstitiedoston lukemisen.

3.4 CSS

CSS eli Cascading Style Sheets on HTML-sivulle annettava tyyliohje, joka määrittää, miten sivun elementit näytetään. HTML-sivun CSS määritykset määräävät, minne HTML-elementit asetellaan ja miltä ne näyttävät. Työssä määriteltiin CSS:llä esim. taulukkojen tekstien fonttia ja kokoa sekä taulukon väritys.

4 JQUERY JA SEN KÄYTTÖ

jQuery on JavaScript-kirjasto, jonka tarkoitus on yksinkertaistaa HTML-dokumentin läpikäymistä ja muokkaamista. jQuerya kutsutaan ns. DOM-crawleriksi. Sillä tarkoittaa sitä, että se pääsee käsiksi kaikkiin HTML-dokumentissa ilmestyviin objekteihin helposti. Avainsana on helposti, sillä puhtaassa JavaScriptissä erilaisten elementtien haku tapahtuu melko kömpelösti.

jQuery on koodattu JavaScriptillä, mikä tarkoittaa, että sen kanssa ei ole yhteensopivuusongelmia JavaScriptin kanssa. jQueryn käyttöönotto on myös hyvin suoraviivaista.

jQueryn suosion taustalla on juuri se, kuinka näppärästi sillä pääsee muokkaamaan DOM:ssa olevia objekteja ja kuinka kätevästi sillä saa liitettyä erilaisia tapahtumia löytyviin objekteihin.

4.1 jQueryn synty

jQuery sai alkunsa heinäkuussa 2005, kun John Resig tutki Behaviour-nimisen JavaScript-lisäosan koodia ja mietti, että se voisi olla sujuvampi ja virtaviivaisempi. Hänen mielestään Behaviour oli liian monimutkaisen näköistä ja ajaisi pois harrastelijakoodaajat. Resig muotoili blogissaan, miten Behaviouria voisi parantaa ja miten tiettyjä asioita voisi tehdä helpommin, ja varsinkin usein käytetyille ominaisuuksille pitäisi olla helppokäyttöiset käskyt.

14. tammikuuta 2006 jQuery esiteltiin BarCampNYC-tapahtumassa ja Digg ja Delicio.us-sivustoilla ilmoitus nousi etusivulle. 25. tammikuuta julkaistiin ensimmäinen kolmannen osapuolen jQuery-plugin ja 30. kesäkuuta julkaistiin jQuery 1.0-alphaversio.

Muita merkittäviä merkkipaaluja on esimerkiksi 2008, kun Microsoft ja Nokia ilmoittivat aloittavansa käyttää jQuerya sisäisessä kehityksessä, sekä 2009, kun jQuery sai .Net Magazine-palkinnon parhaasta avoimen lähdekoodin ohjelmasta. (jQuery Foundation 2014)

4.2 jQueryn käyttöönotto

jQueryn käyttöönotossa on hyvä muistaa muutama asia. Harjoitustyössäni tätä ei tarvinnut miettiä, koska kyseessä oli työpöytäsovellus, mutta jos aiot tehdä internetsivun, jossa käytetään jQuerya, on harkittava muutamaa asiaa. Ensin täytyy muistaa, että kyseessä on JavaScript-kirjasto, jonka jokainen sivulla kävijä lataa koneelleen. jQuerystä tarjotaan pakattua ja ei-pakattua versiota, joista ensimmäinen on suorituskyvyltään heikompi mutta kooltaan paljon pienempi ja tämän vuoksi parempi valinta internetsivuille. jQuerya saa myös CDN-versioina, jolloin tiedosto ei sijaitse omalla serverillä vaan se haetaan CDN:stä. Tällöin on syytä muistaa pitää oma koodi ajan tasalla, koska aina uuden version ilmestyessä vanha koodi saattaa deprekoitua ja sivusto hajota.

Työpöytäsovellusta tehdessä pystyttiin ottamaan pakkaamaton versio ja käyttämään sitä, koska ei tarvinnut huolehtia kaistan käytöstä. HTML-dokumentille pitää kertoa, missä jQuery-tiedosto sijaitsee. Tehdyssä sovelluksessa jQuery sijaitsi Resources-kansiossa, joka oli samassa kansiossa HTML-dokumentin kanssa. jQueryn sijainti kirjoitetaan omien script-tagien sisään. Kuvan 1 esimerkki on otettu omasta työstä.

```
<script src="Resources/jquery-1.11.2.js"> </script>
```

Kuva 1. Pakkaamaton jQuery Resources- kansiosta

Tilanteessa, jossa jQuery-tiedosto on samassa kansiossa HTML-dokumentin kanssa, osoitteesta tiputetaan ”Resources/”-osa pois. Minimoidussa jQuery-versiossa on yleensä sana ”min” osoittamassa, että kyseessä on minimoitu versio, jolloin kuvan 1 loppuosa olisi muotoa jquery-1.11.2.min.js. Sinänsä tiedoston nimellä ei ole väliä, kunhan polku osoittaa oikeaan tiedostoon.

CDN-versioissa tiedostot haetaan internetistä sivun latauksen yhteydessä ja tällöin osoite osoitetaan internetosoitteeseen. Osoite vaihtelee sen perusteella, miltä CDN-palveluntarjoajalta sen käy hakemassa.

jQuerystä on tarjolla kahta eri versiota, eli 1.x-versiota ja 2.x-versiota. Näillä on erona vain se että 2.x versio ei tue Internet Explorer 6-, 7- tai 8-selainta laisinkaan. (jQuery 2015.)

jQueryn syntaksi eroaa JavaScriptin syntaksista melkoisesti ja ainakin alkuun vaikutti melko sekavalta. jQuery-lause alkaa aina dollari ($) merkillä tai sanalla ”jquery” mutta jälkimmäistä ei juurikaan käytetä. Tämän jälkeen tulee suluissa etsittävät elementit ja sen jälkeen, pisteellä erotettuna, mitä niille tehdään.

4.3 Miksi jQuerya kannattaa käyttää

JavaScriptin ja jQueryn suurin ero on siinä miten päästään käsiksi DOM:ssa oleviin elementteihin. JavaScript tarjoaa komentoja, joilla elementtejä kutsutaan kuten getElementById() ja getElementsByTagName(). GetElementById() palauttaa elementin, jonka nimi määritellään sulkujen sisällä ja getElementsByTagName() palauttaa taulukon elementeistä, jotka jakavat tagin nimen, joka tulee sulkuihin. jQuery hakee elementit DOM:sta ns. valitsimilla, jotka hakevat halutun elementin vaikka elementin tyypin perusteella, mutta näihin valitsimiin liitetään myös tapahtuma, joka elementille tai elementeille tehdään. Valitsin käy kaikki löytämänsä elementit läpi ja niille kaikille tehdään määritelty asia. Tietenkin valitsimen voi määritellä myös niin, että se löytää vain yhden elementin. Valitsimen käyttäminen ja siihen liittyvän funktion suora liittäminen tekevät koodista paljon selkeämpää ja lyhempää, mikä taas on tehokkaampaa, ainakin koodaajan näkökulmasta.

Valitsimen käyttäminen on paljon parempi tapa käsitellä elementtejä kuin käyttää JavaScriptin elementin valintametodeja. Valitsimen automaattinen silmukkarakenne säästää myös päänvaivaa kun ei tarvitse rakentaa silmukkaa käsin ja laskeskella, montako elementtiä valittiin.

Valitsimen avulla on myös helppo lisätä tapahtumat elementteihin esimerkiksi, mitä tapahtuu kun elementtiä klikkaa. Tämän tapahtuman voi lisätä helposti, mihin tahansa elementtiin eikä vain nappuloihin. Tämä taas tuo uusia ulottuvuuksia sivujen interaktiivisuuteen.

4.4 Esimerkkejä jQueryn käytöstä

Nyrkkisääntönä voisi sanoa, että jos käsitellään UI-elementtejä, kannattaa käyttää jQuerya. Tällöin usein pääsee vähemmällä kirjoittamisella, ja koodista tulee kevyempää ja selkeämpää. Esimerkkinä JavaScript- ja jQuery-koodit ns. seepraraitojen tekemisestä table-elementtiin (taulukko). Molemmissa esimerkeissä on oltava CSS:llä määritelty luokka ”.odd”. Kuvan 2 esimerkki on tehty puhtaalla JavaScriptillä.

```
function alternateRows(){
// change row color for every other row for clarity
var rows = table.getElementsByTagName('tr');
var rCount = rows.length;

    for(i = 1; i < rows.length; i++){

   if(i % 2 == 0){
        rows[i].className = "even";
      }
      else{
        rows[i].className = "odd";
      }
    }
}
```

Kuva 2. Seepraraidat JavaScriptillä

Kuvassa 3 sama asia toteutetaan jQuerylla.

```
function alternateRows(){
$("tr:odd").addClass("odd");
}
```

Kuva 3. Seepraraidat jQuerylla

Kuten esimerkistä näkee, on koodin määrässä paljon eroa. jQuerylle ei tarvitse kertoa, mistä taulukosta tr-elementit (rivit) haetaan ja kuinka monta riviä taulukossa on. jQuery hakee kaikki tr-elementit ja vaihtaa parittomien tr-elementtien luokaksi ”.odd”.

jQuery koodissa $(”tr:odd”) tarkoittaa, että otetaan kaikki tr-tagin elementeistä parittoman indeksin omaavat ja addClass(”odd”) lisää näihin CSS-luokan ”.odd” ominaisuudet. $(”tr:odd”)-osaa koodista kutsutaan selektoriksi eli valitsimeksi, koska sen avulla elementit valitaan. Esimerkissä ongelmia voisi tulla, jos sivulla on useampi taulukko ja osaan halutaan raidoitus ja osaan ei. Tällöin voidaan kohdistaa raidoitus tiettyyn taulukkoon antamalla taulukon ID-attribuutti, josta etsitään kaikki tr-elementit. Esimerkissä taulukon ID oli ”outputTable” ja silloin koodin alku näyttäisi tältä $(”table#outputTable tr:odd”). ”Table#outputTable” tarkoittaa, että otetaan table-elementti, jonka ID on ”outputTable”, jos valitaan sen kaikki tr-elementit.

JavaScriptin kanssa joutuu käyttämään kömpelöitä .getElementsBy-komentoja ja rakentamaan niille usein vielä silmukan, joka tekee haetuille elementeille tietyn asian. jQueryssa silmukka on ikään kuin rakennettu sisään valitsimeen eli se hakee kaikki tagia vastaavat elementit ja suorittaa jokaiselle halutun asian.

Työssä oli myös ongelma, jossa turhia elementtejä piti poistaa näkyvistä. Myös tässä tapauksessa jQuery tarjosi helpon ratkaisun. Tarkoituksena oli siis poistaa ylimääräiset napit, joita ei enää tarvittu. Tällä kertaa ei riittänyt, että kerätään kaikki elementit ja tehdään niille yksi ja sama operaatio, vaan piti erotella poistettavat nappulat omaksi joukoksi. Omassa työssä nappuloiden nimet tallennettiin taulukkoon ja nappeja poistettaessa tiedettiin aina, minkä taulukon nappeja ei enää tarvittu ja taulukoista sai nappien nimet, jotka poistetaan. Kuvan 4 koodi on otettu omasta työstä.

```
function deletebutn (taso){
        //deletes unnecessary buttons in case the user goes back up in the tree
        var pituus = taso.length;
         $(":input[type=submit]").each(function(){
           for(i = 0; i < pituus; i++){
         if($(this).val() == taso[i])
            $(this).remove();
                 }
         });
       }
```

Kuva 4. Nappien poisto-funktio.

Kyseisessä koodipätkässä etsitään kaikki napit ja vertaillaan niitä taulukossa ”taso” oleviin arvoihin. Ehdon täyttyessä kyseinen nappula poistetaan. Aina, kun poistoa tehdään, pitää tietää, minkä taulukon nappeja poistetaan. Kuvan 4 esimerkissä taulukko annettiin manuaalisesti koodissa, funktion parametrina, mikä sopi, koska ylimääräiset napit olivat aina tiedossa.

Funktio kuvaa hyvin, kuinka JavaScript ja jQuery toimivat hyvin yhdessä. Taulukon läpikäyvä for-silmukka on puhdasta JavaScriptiä ja sen sisällä tapahtuva ehtolause taas jQuerya.

Selvennetään ehtolauseen sisällä näkyvää $(this)-metodia. $(this) palauttaa aina valitun objektin, eli esimerkissä napit, jotka valitaan $(”:input [type = submit]”) komennolla. Tämä tehdään kaikille HTML-sivulta löytyville napeille ja $(this) palauttaa aina sen napin, jota tällä hetkellä käsitellään. .val() on metodi, joka palauttaa valitun elementin arvon eli tässä tapauksessa napin sisältämän tekstin, joka taas on myös napin nimi, ja tätä verrataan taulukosta löytyviin arvoihin.

jQueryssa voidaan hyödyntää ns. nimettömiä funktioita. Kuvan 4 esimerkin .each(function) tarkoittaa, että jokaiselle valitulle objektille, eli tässä tapauksessa napille, tehdään lauseen jälkeen kaarisulkujen sisään tehty asia. Ero tavallisiin funktioihin on se, että funktiolla nimensä mukaan ei ole nimeä. Se on vain function(). Tämäkin säästää kirjoittamiselta, mutta samalla on syytä varoa, ettei omista nimettömistä funktioista kasva suurta nimettömien funktioiden metsää, jota on mahdotonta ylläpitää. Itse asiassa jQueryn oma Learning Center-sivu (2015) varoittaa, että nimettömiä funktioita ei tulisi käyttää ollenkaan juuri ylläpitämisen ja koodin testaamisen vaikeuden takia. Tätä varoitusta ei tosin näe juuri noudatettavan vaan internet on pullollaan esimerkkejä, joissa viljellään paljonkin nimettömiä funktioita.

4.4.1 Nimettömien funktioiden välttäminen

Nimettömät funktiot eivät itsessään ole kamala asia, jota tulisi välttää kaikin keinoin vaan se, miten niitä käytetään. Nimettömien funktioiden sisällä usein näkee toisen funktiomäärittelyn ja nämä sisäkkäiset funktiomäärittelyt on se asia, jota on syytä välttää kaikin keinoin. Sisäkkäiset funktiot

yleensä aiheuttavat suorituskykyongelmia johtuen tavasta, jolla JavaScript käsittelee funktioita. Sisäisiä funktioita kutsuttaessa JavaScript joutuu antamaan funktiolle arvon ja varaamaan sille tilaa muistista. Funktion suorittamisen jälkeen arvo tuhotaan, mikä taas kuluttaa suoritusaikaa. (Jeremy McPeak 2011.) Kuten kuvassa 4 näkyy, nimetön funktio suoritetaan toisen sisällä, jolloin funktion määrittely tehdään vain funktiota kutsuttaessa. Nyt jos tämän sisällä määriteltäisiin ja sitten suoritettaisiin toinen funktio, suoritus olisi tehoton ja sama tehoton suoritus tehtäisiin jokaiselle nappulalle.

Nimelliset funktiot luodaan yleensä sitä varten, että niitä tullaan kutsumaan useaan otteeseen, kun taas nimettömän funktion uusiokäyttö vaatii kaikkea funktion sisällä olevan tekstin kopiointia uuteen paikkaan. Kopioinnissa on kuitenkin myös se riski, että virheet monistuvat ja sitten pitää etsiä kaikki kopiot tästä funktiosta ja korjata nekin. Puhumattakaan siitä, että tällaisten kertautuvien virheiden yhteisvaikutukset tekevät virheen paikantamisesta tarpeettoman hankalaa. Tätä kutsutaan myös nimellä DRY eli Don't Repeat Yourself, eli jos toistat itseäsi ja kirjoitat samoja funktioita yhä uudestaan ja uudestaan, teet jotain väärin.

Ohjelmoidessa tulee välttää tilanteita, joissa funktioita määritellään toisen sisällä. JavaScriptissä tämä johtaa pahasti heikentyneeseen suorituskykyyn. Pitää olla tarkkana myös jQueryn automaattisten silmukoiden kanssa, sillä niissä on vaarana se, että etsittävät tagit esiintyvät sivustolla todella monta kertaa ja tällaiseen silmukkaan on usein lyöty perään nimetön funktio, joten on tärkeää, että suoritus ei syö ylimääräistä aikaa.

Kuvan 4 tapauksessa for-silmukka ja sen sisällä oleva ehtolause luodaan ja käydään läpi uudestaan jokaiselle nappulalle erikseen. Onneksi ohjelmassa olevia nappuloita oli pahimmassakin tapauksessa vain vähän. Tilanne voisi olla huomattavasti pahempi, jos sisäkkäin olevia nimettömiä funktioita olisi monia. Myös tilanne, jossa nappuloita olisi ollut suurempi määrä, suorituskyky olisi voinut kärsiä. Etenkin jos mukana olisi myös nappeja, joita ei tarvitsisi edes testata. Aina sisäisiä funktioita käytettäessä kannattaa miettiä, kuinka monta kertaa se pahimmassa tapauksessa tullaan suorittamaan. Sisäkkäin olevat funktiomäärittelyt ovat selvä ohjelmointivirhe, jota tulee ehdottomasti välttää. Se syö suorituskykyä eikä salli koodin uudelleen käyttöä.

4.4.2 Optimointikikkoja

jQueryn omat dokumentit kertovat, miten jQuerysta saa enemmän suorituskykyä irti, ja ne on hyvä tiedostaa ja pitää mielessä. Tässä esitellään perusjutut, joihin tulee törmättyä pienemmissäkin ohjelmissa ja osa toimii ihan JavaScriptinkin kanssa. jQueryn dokumentoinnissa (2015) on lueteltu seuraavia kikkoja.

Taulukoita käytäessä läpi, kannattaa tallentaa taulukon pituus muuttujaan ja käyttäää muuttujaa käydessä taulukkoa läpi, ettei tarvitse joka kerta laskea taulukon pituutta

Älä hae elementtejä, joita ei ole. Kuulostaa yksinkertaiselta, mutta näitä saattaa joka tapauksessa livahtaa läpi ja aikaa kuluu hukkaan kun etsitään elementtejä turhaan.

ID:llä etsiminen on nopein jQuery-haku, koska se käyttää selaimen natiivia document.getElementById-metodia.

Valitsimissa kannatta käyttää pelkkää tagia tai .luokka-hakua, jolla löytyy haluttu elementti, jonka sisältä etsitään tarkemmin, ja jos on tarpeen, tarkennetaan tag.luokka-haulla. Eli epätarkempi alussa ja tarkempi lopussa. Liian tarkaksi ei kannata mennä, sillä se taas hidastaa hakua ja maksimissaan kahden hakulauseen käyttö peräkkäin on hyvä perussääntö.

Litteä DOM-rakenne tekee hakemisesta nopeampaa eli ei liikaa sisäkkäisiä ja taas sisäkkäisiä elementtejä.

4.4.3 $(Document).ready()-metodi

jQuery tarjoaa metodin, joka mahdollistaa jQuery-koodin suorittamisen vasta sen jälkeen, kun sivun koko DOM on latautunut. Tämä mahdollistaa turvallisen DOM:n muokkaamisen, kun voi olla varma, että kaikki tarvittavat tiedot ovat latautuneet eikä yritetä muokata olemattomia tai vain osin latautuneita elementtejä. Tämän metodin käyttöön tulee varmasti törmäämään lukiessaan jQuery-esimerkkejä. Metodin voi kirjoittaa kahdella tavalla:

- $(document).ready(function() { koodi tulee tähän });
- $(function() { koodi tulee tähän });

Molemmat tekevät saman asian ja molempiin tulee törmäämään. Varsinkin jälkimmäinen voi aiheuttaa hämmennystä, jos tällaisen metodin olemassaolosta ei tiedä.(jQuery 2015.)

Document.ready käynnistyy heti kun koko DOM on latautunut. Se käynnistyy ennen kuin koko sivu on latautunut. Tämä mahdollistaa sivun muokkaamisen ennen kuin käyttäjä edes näkee sivua ja täten mahdollistaa erinäisiä ulkonäköön liittyviä kikkoja. Pelkkä DOM:n lataamisen odottaminen erottaa metodin JavaScriptin vastaavasta window.onload-metodista, jossa koko sivun kuvineen päivineen odotetaan lataavan ennen kuin mitään tapahtuu.

Document.ready-metodia käytetään usein, kun omat skriptitiedostot ovat ulkoisessa tiedostossa. Usein tämä tiedosto on oman document.ready-metodin sisällä ja täten käytettävissä heti ensimmäisellä turvallisella hetkellä. Täytyy vain muistaa, että CSS-tiedosto määritetään ennen jQuery-tiedoston lataamista ja tämän jälkeen oma ulkoinen JavaScript-tiedosto document.ready-koodipalikan sisällä.(Swedberg 2006.)

4.5 Valitsimet

Valitsimet ovat keino, jolla jQuery etsii tahdotut elementit DOM:sta ja mahdollistaa näiden manipuloinnin. Tämä on erittäin kätevää jos manipuloitavana on monta elementtiä.

Elementtien hakeminen kannattaa ottaa huomioon suunnittelun aikaisessa vaiheessa, sillä eri elementit, joita tullaan muokkaamaan, kannattaa erottaa muista elementeistä jollain tavalla. Kannattaa myös muistaa, että HTML-kielessä elementeille voi antaa mitä tahansa, myös itse keksittyjä, attribuutteja eikä siitä koidu minkäänlaisia harmeja.

Katsotaan, miten valitsimet muodostetaan. Eli ensiksi valitaan käsiteltävä elementti ja se voidaan valita esimerkiksi nimen tai tyypin mukaan. Valitsimessa pelkkä elementin tyyppi valitsee kaikki kyseiset elementit, mutta valitsimessa voi rajata useamman kriteerin mukaan vaikka kaikki tietyn tyypin ja nimessä olevan tekstipätkän perusteella. Esimerkiksi kaikkien div-elementtien valitsin olisi $(”div”) (jQuery 2015). Kaikkien eri HTML-elementtien valitseminen toimii samalla lailla eli heittomerkkien sisään elementin tyyppi.

4.5.1 Perusvalitsimet

Perusvalitsimiksi kutsutaan valitsimia, joilla haetaan kokonaisia ryhmiä elementtejä ilman sen kummempia karsimiskriteerejä. Perusvalitsimien käyttö on hyvin suoraviivaista. Perusvalitsimet käydään läpi taulukossa 1.

Taulukko 1. Perusvalitsimet

Valitsimen nimi	Merkki	Selitys
Kaikkien elementtien valitseminen	(*)	Valitsee nimensä mukaan kaikki sivulla olevat elementit
Luokka-valitsin	(.LuokanNimi)	Valitsee kaikki elementit, joille on määritelty luokka LuokanNimi
Elementti-valitsin	(”Elementti”)	Löytää kaikki Elementin tyyppiset elementit. Esim. ”div”
ID-valitsin	(”#idArvo”)	Palauttaa vain yhden tai ei yhtään elementtiä, jolle on annettu tämä id-arvo

Perusvalitsimia voi myös pistää pötköksi peräjälkeen pilkulla erotettuna, jolloin etsitään kaikki joukkoon kuuluvat elementit. Esim. (”div, .omaLuokka”), jolloin löytyy kaikki div-elementit ja elementit, joille on annettu luokka omaLuokka. (jQuery 2015.)

Ominaisuuden ID pitää olla jokaisella elementillä uniikki ja yhden ID:n pitäisi esiintyä koko dokumentissa vain kerran. jQuery tukee etsimistä myös ID:n perusteella, mutta palauttaa vain yhden, ensimmäisen osuman, tai ei yhtään elementtiä siinä tapauksessa, että sitä ei löydy. Syyksi sanotaan, että jos useammalla elementillä on sama ID, on dokumentti epäkelpo. (jQuery 2015.)

4.5.2 Attribuuttivalitsimet

Tarkempi rajaus taas saadaan yhdistämällä elementtiin erilaisia suodattimia, joiden avulla saa vaikka div-elementit joille on annettu tietty nimi attribuutti. Esimerkiksi $("div[name*=’oma’]"). Tässä hakasulkujen sisään tulee attribuutti, jolla haetaan eli name, attribuuttia seuraava tähti (*) merkitsee, miten haetaan. Tähti tarkoittaa, että ”elementti sisältää”, eli kaikki elementit, joiden nimi sisältää tekstipätkän, jonka arvo annetaan yhtäsuuruusmerkin (=) jälkeen. Hakasulkuja ennen sijaitseva ”div” taas kertoo, minkälaisia elementtejä etsitään ja jos dokumentissa on jonkin muun tyypin elementti, jolla on sama nimi, tätä elementtiä ei oteta huomioon. Tähden kaltaiset erityismerkit esitetään taulukossa 2.

Taulukko 2. Valitsimen lisämerkit

Valitsimen kuvaus	Merkki	Selitys
Attribuutti sisältää etuliitteen	|	Attribuutti vastaa tai alkaa annetulla merkkijonolla ja sitä voi seurata -
Attribuutin arvo sisältää	*	Elementin attribuutti sisältää annetun merkkijonon.
Attribuutin arvo sisältää sanan	~	Attribuutti sisältää annetun merkkijonon rajattuna välilyönnein.
Attribuutin arvo loppuu	$	Elementin attribuutti loppuu annettuun merkkijonoon.
Attribuutti ei vastaa arvoa	!	Elementin attribuutit, jotka ei vastaa annettua arvoa, ja myös elementit, joille ei ole määritelty annettua attribuuttia
Attribuutin arvo alkaa	^	Elementin attribuutin arvo alkaa annetulla merkkijonolla

Lisäksi voisi vielä mainita erikoistapaukset eli pelkkä $(”elementti[name = ’oma’]”) on ns. perinteinen haku, jolla haetaan vain ne elementit, joilla on annettu attribuutti ja sillä tismalleen tämä arvo. Pelkän attribuutin antaminen eli $(”elementti[name]”) etsii kaikki elementit, joilla on annettu attribuutti mutta sen arvosta ei välitetä. Sitten on vielä monen kriteerin perusteella valittujen elementtien haku eli esim. $(”elementti[name = ’oma’] [name2 = ’oma2’]”) hakee elementit, joille on määritelty name ja name2 ja arvot ovat oma ja oma2. (jQuery 2015.)

Heittomerkkien kanssa pitää olla tarkkana, sillä pitää muistaa niiden oikea käyttötapa. Oikeita tapoja on käyttää yksinkertaisten heittomerkkien(’) sisällä kaksinkertaisia merkkejä(”) tai kaksinkertaisten heittomerkkien sisällä yksinkertaisia merkkejä. Esimerkiksi $(”div[name=’oma’]”) ja

$(’div[name=”oma”]’) ovat oikeaoppisia. Voi myös käyttää ns. escape merkkiä eli ”\”-merkkiä ennen heittomerkkejä jolloin voi käyttää samanlaisia heittomerkkejä sisäkkäin esim. $(”div[name=\”oma\”]”) ja $(’div[name=\’oma\’]’) käyvät myös. (jQuery 2015.)

4.5.3 Sisältövalitsimet

Sisältövalitsimilla tarkennetaan perusvalitsimien käyttöä ja annetaan mahdollisuus valita elementtejä sen perusteella, mitä ne sisältävät. Sisältövalitsimilla voi tarkastella elementin sisältävää tekstiä tai elementin sisällä olevaa toista elementtiä. Voi myös valita elementtejä sen perusteella, puuttuvatko niiltä lapsisolmut eli onko kyseessä tyhjä elementti, vai onko elementti isäsolmu jollekin muulle elementille.

Sisältövalitsimien käyttö on melko yksinkertaista. Ensin valitaan elementti, josta sisältö etsitään ja sen perään kirjoitetaan jokin alla esitellyistä komennoista. Esim. $(”div:contains(’tekstipätkä’)”) valitsee div elementit, joiden sisällä on sisältö ”tekstipätkä”. Sisältövalitsimet esitellään taulukossa 3.

Taulukko 3. Sisältövalitsimet

Valitsimen kuvaus	Komento	Selitys
Pitää sisällään tekstin	(”:contains(’text’)”)	Valitsee elementin jonka sisällä on määritelty tekstipätkä (merkkikokoriippuvainen)
Tyhjä elementti	(”:empty”)	Valitsee elementit, joilla ei ole lapsielementtiä
Pitää sisällään elementin	(”:has(Elementti)”)	Valitsee elementit, jotka pitävät sisällään ”Elementti”-elementin
Pitää sisällään joko toisen elementin tai tekstin	(”:parent)	Valitsee elementit, joilla on yksi tai useampi lapsisolmu (voi olla tekstiä tai elementti)

Tyhjiä elementtejä etsittäessä kannattaa muistaa, että jotkin HTML-elementit ovat aina lapsettomia, kuten
 (rivinvaihto) ja <img> (kuva). (jQuery 2015.)

4.5.4 Perusvalitsimien suodattimia

Perusvalitsimet itsessään ovat hyvin ylimalkaisia eivätkä kovinkaan tarkkoja varsinkaan dokumenttien koon kasvaessa. Perusvalitsimille on luotu erilaisia tarkennusmahdollisuuksia ja tässä käydään läpi joitain niistä, jotka kokisin hyödylliseksi esimerkiksi käyttöliittymien kannalta.

Yhteistä kaikille suodattimille on se, että ne valitsevat elementtejä niiden järjestysluvun tai tyypin perusteella. Järjestysluku määräytyy sen mukaan, missä kohtaa dokumenttia kyseinen elementti ilmestyy ja onko sitä ennen jo ollut samanlaisia elementtejä. Myös parilliset ja parittomat elementit löytyvät ja näissä käytetään ns. nollapohjaista indeksointia eli ensimmäisen elementin järjestysluku on nolla. Parillisia ja parittomia elementtejä valittaessa on hyvä tietää, että parilliset elementit ovat, nollapohjaisesta indeksoinnista johtuen, ensimmäinen, kolmas, viides jne., kun taas parittomat elementit ovat toinen, neljäs, kuudes, jne., eli juuri toisinpäin kuin tavallisesti. (jQuery 2015.)

Nämä suodattimet toimivat siis perusvalitsimien kanssa esim. $(”div:first”) palauttaa koko dokumentin ensimmäisen div elementin ja $(”.oma:last”) koko dokumentin viimeisen ”oma” luokan omaavan elementin

Indeksipohjaisia valitsimia on kätevintä käyttää muokkaamaan mm. taulukkojen ulkonäköä, mutta en näe syytä, miksei niillä voisi muokata myös tavallisia sivustoja, kunhan elementtien järjestysluvut ovat tiedossa ja niiden järjestys ei tule missään vaiheessa muuttumaan. Taulukossa 4 käydään läpi osa suodatuskomennoista.

Taulukko 4. Osa perusvalitsimien suodattimista

Valitsimen kuvaus	Komento	Selitys
Parittomat-valitsin	:odd	Hakee parittomat elementit
Parilliset-valitsin	:even	Hakee parilliset elementit
Ensimmäisen valitsin	:first	Palauttaa ensimmäisen etsityn tyypin elementin
Viimeisen valitsin	:last	Palauttaa viimeisen etsityn tyypin elementin
Otsikko-valitsin	:header	Palauttaa kaikki otsikkoelementit h1-h6
Käänteinen valitsin	:not()	Palauttaa kaikki elementit, jotka eivät vastaa haettua elementtiä

4.5.5 Piilotetut ja näkyvät elementit

jQuerylla on myös helppoa etsiä piilotettuja tai näkyviä elementtejä, mutta niissä on omat erikoissääntönsä, mikä katsotaan näkyväksi ja mikä piilotetuksi elementiksi.

Elementti on piilotettu, jos
- CSS:n määritelty display-arvo (näkyvyys) on ”none”.
- Elementti on form-elementin (lomake) osa, jonka type=”hidden”.
- Leveydelle ja korkeudelle (width ja height) määritelty arvo on 0.
- Elementti on näkymättömän elementin sisällä.

Muuten elementti on näkyvä. Huolimatta siitä, että elementin visibility olisi ”hidden”, tai elementin opacity olisi 0, lasketaan ne näkyviksi. Syy tähän on, että vaikka paljaalle silmälle ne ovat näkymättömissä, vievät ne silti tilaa itse dokumentista. Itse komennot löytyvät taulukosta 5. (jQuery 2015.)

Taulukko 5. Komennot näkyvyyden perusteella suodattamiseen

Valitsimen kuvaus	Komento	Kuvaus
Näkyvien elementtien valitsin	:visible	Hakee valitut, näkyvissä olevat, elementit
Piilotettujen elementtien valitsin	:hidden	Hakee valitut, piilotetut, elementit

4.6 CSS-ominaisuudet

Kun valitsimet ovat hallussa, olisi hyvä tietää tarkemmin, mitä niillä tehdään eli miksi erinäisiä elementtejä valitaan. Tarkoituksena on lisätä valittuihin elementteihin tapahtumia, jotka vaihtelevat elementtien ulkonäön muuttamisesta hiiren napin klikkaamisesta käynnistyvään funktioon. jQuery ei ole pakollinen näiden toimintojen tekemiseen, mutta yhdessä valitsimien kanssa näiden toimintojen lisääminen käy huomattavasti helpommin kuin pelkästään HTML:llä tai JavaScriptillä.

Tapahtumat kirjoitetaan valitsimen perään ja tapahtuma vaikuttaa jokaiseen valitsimen palauttamaan elementtiin. Katsotaan ensin, miten CSS ominaisuuksia saa muutettua valitsemalleen elementtijoukolle.

Valitsinosa on ensin, sen jälkeen .css (validia CSS:ää). Eli jos sivulle on määritelty div-elementti , saa sen CSS-ominaisuuksia muutettua esim. koodipätkällä $("div[name=’oma’]").css("border", "5 px solid red"). Tämä koodipätkä lisää div-elementin ympärille punaisen rajauksen. Valitsimen jälkeinen komento muuttaa määriteltyä CSS-arvoa, jos sellainen jo on. Siina tapauksessa, jossa CSS-arvoa ei ole, arvo lisätään. Komennolla voi myös poistaa annetun arvon muuttamalla halutun arvon tyhjäksi. Tällöin elementin tyyli palaa takaisin siihen, mikä sille oli määrätty CSS-määrittelyssä. Sillä ei siis voi poistaa itse CSS-määrittelyä. (jQuery 2015.)

Yhdistettynä erilaisiin tapahtumiin CSS-ominaisuuksien komennoilla saakin luotua näppärästi yksinkertaisia koodipätkiä, joita voi käyttää vaikka luomaan erikoistehosteita tekstille, kun sen päälle vie hiirenkursorin (ns. mouseover-tapahtuma). Näistä hiiritapahtumista kerrotaan lisää myöhemmässä osassa.

Komennolla .css saa myös kysyttyä elementille jo annettuja CSS-arvoja. Käyttäjä ei näillä arvoilla tietenkään tee mitään, mutta niillä voi saada CSS-arvon ja sen jälkeen muokata vaikkapa tekstin kokoa napin painalluksella ja muutenkin se mahdollistaa elementtien manipuloinnin lennosta. (jQuery 2015.)

Yksi tehokkaimmista työkaluista CSS:n muokkaamiseen on .addClass-, .removeClass- ja .toggleClass-komennot, joilla nimensä mukaisesti lisätään, poistetaan ja kytketään päälle tai pois CSS-luokkia elementeiltä. CSS-komennot käydään läpi taulukossa 6. (jQuery 2015.)

Taulukko 6. CSS-luokkien manipulointi komennot

Nimi	Komento	Selitys
Lisää luokka	.addClass()	Lisää elementille määritellyn luokan
Tarkista luokka	.hasClass()	Tarkastaa, onko elementillä määritettyä luokkaa
Poista luokka	.removeClass()	Poistaa elementiltä määritellyn luokan
Luokka vaihdin	.toggleClass()	Kytkee määritellyn luokan päälle tai pois päältä

Lisää luokka-komennolla elementille lisätään sulkujen sisällä annettu CSS-luokka. Luokan pitää olla ennestään määriteltynä.

Poista luokka-komennolla elementiltä poistetaan sulkujen sisällä määritelty CSS-luokka.

Luokka vaihdin-komento vaihtaa sulkujen sisällä määritellyn CSS-luokan päälle, jos se ei jo ennestään ole päällä tai pois päältä siinä tapauksessa, että se on ennestään päällä.

Tarkista luokka-komento tarkistaa, onko valitulla elementillä määritelty luokka jo päällä. Palauttaa arvon tosi (true) tai epätosi (false).

Komennoille voi antaa useamman luokan kerrallaan ja ne erotetaan toisistaan pelkillä välilyönneillä. Luokkien lisäämisessä ja poistamisessa kannattaa olla varovainen, sillä eri luokkien lisääminen päällekkäin johtaa siihen, että päällekkäin menevissä CSS-arvoissa alempana CSS-määrityksessä oleva arvo jää päälle. Kannattaa mieluummin ketjuttaa poisto ja lisäys samaan lauseeseen ja poistaa vanha luokka ensin ja sen jälkeen lisätä uusi, vaikka näin: $("div[name='oma']").removeClass("vanha").addClass("uusi"). On hyvää pitää mielessä, että elementin luokkien poistaminen saattaa haitata myöhempiä jQueryn valitsinhakuja, jos ne etsivät elementtejä luokkien perusteella. Tilanteessa, jossa luokkia vaihdellaan useammin edestakaisin, olisi parempi käyttää toggleClass() komentoa. (jQuery 2015.)

4.7 jQueryn animaatiot

jQuery tarjoaa myös sisäänrakennettuna animaatioita. Kustomoitujen animaatioiden lisäksi tarjotaan haihdutus ja liu’utus. Haihduttamisessa määritellyt elementit katoavat tai ilmestyvät määritettyä vauhtia ja liu’utuksessa elementit liukuvat esiin tai piiloon. Liu’utuksessa elementti ilmestyy aina ylhäältä alaspäin ja katoaa alhaalta ylöspäin. Haihdutus ja liu’utus on

helppo lisätä koodiin ja niillä saa aikaan ammattimaista jälkeä pienellä vaivalla. Pitää tietenkin muistaa, että joskus vähemmän on enemmän, etenkin animaatioiden keston kanssa. Kannattaa miettiä, kauanko käyttäjä on valmis odottamaan animaatioiden valmistumista. Lyhytkestoisetkin animaatiot voivat käydä ärsyttäviksi, jos niitä laittaa joka paikkaan ja pitkäkestoiset animaatiot saavat pahimmillaan käyttäjän arvailemaan sovelluksen toimivuutta.

4.7.1 Haihdutus

Käydään ensin läpi haihdutukseen liittyvät komennot. Haihdutuksessa elementit saadaan täysin katoamaan tai muuttumaan vain osittain läpinäkyväksi. Taulukossa 7 käydään läpi haihdutuskomennot. (jQuery 2015.)

Taulukko 7. Haihdutuskomennot

Animaation nimi	Komento	Selitys
Ilmestyminen	.fadeIn()	Tuo piilotetun elementin näkyviin
Haihdutus	.fadeOut()	Haihduttaa elementin näkyvistä
Osittainen haihdutus	.fadeTo()	Säätää elementin läpinäkyvyyttä
Haihdutus kytkin	.fadeToggle()	Piilottaa näkyvät tai näyttää piilotetut elementit.

Työssä ei käytetty animaatioita, koska niitä ei koettu tärkeiksi, mutta kuvan 5 esimerkissä lisään animaatiot työn koodiin.

```
function deletebutn (taso){
        //deletes unnecessary buttons in case the user goes back up in the tree
         $(":input[type=submit]").each(function(){

           for(i=0;i<taso.length;i++){

         if($(this).val()==taso[i])
         $(this ).fadeOut( 1000, function() {
            $(this).remove();
       });
    }

  });

}
```

Kuva 5. Animaation lisäys

Kuvassa 5 on lisätty kuvan 4 esimerkkiin animaatio. Ainoa muutos on $(this).fadeOut(1000, function () -rivi. FadeOut lisää haihdutusanimaation nappiin $(this), luku 1000 on animaation kesto millisekunteina ja function määrittää, mitä tapahtuu kun animaatio on suoritettu.

Näin yksinkertaisessa tapauksessa ei kannata laittaa funktiokutsua ja animaation voi silloin tehdä kuvan 6 mukaisesti:

```
function deletebutn (taso){
       //deletes unnecessary buttons in case the user goes back up in the tree
        $(":input[type=submit]").each(function(){

          for(i=0;i<taso.length;i++){

        if($(this).val()==taso[i]){
        $(this).fadeOut( "slow" );
      }
    }

  });

}
```

Kuva 6. Animaation lisäys ilman funktiokutsua

Kuvan 6 esimerkki näyttää hyvin paljon samalta kuin alkuperäinen kuva 4. $(this).remove on vaihdettu $(this).fadeOut(”slow”) komennolla. Nappeja ei tarvitse enää poistaa, koska ne haihtuvat näkyvistä. Jos fadeOut komennon jälkeen olisi remove-komento (poisto), ei animaatio ehtisi valmistua ennen kuin napit jo poistetaan. Kun animaatio tehdään kuten kuvassa 6, ei animaation loppumista odotella.

4.7.2 Liu’utus

Liu’utukselle löytyvät samaa asiaa ajavat komennot kuin haihdutukselle, mutta elementtejä ei saa osittain piiloon. Taulukossa 8 esitellään liu’utuskomennot.

Taulukko 8. Liu’utuskomennot

Animaation nimi	Komento	Selitys
Liu’utus esiin	.slideDown()	Liu’uttaa valitut elementit esiin
Liu’tus kytkin	.slideToggle()	Liu’utta näkyvät elementit piiloon tai piilotetut näkyviin
Liu’tus piiloon	.slideUp()	Liu’uttaa näkyvän elementin piiloon

Liu’utus lisätään samalla periaatteella. Elementtien piilottaminen saadaan aikaan komennolla slideUp ja elementit tulevat näkyviin komennolla slideDown. SlideToggle puolestaan näyttää piilotetut elementit ja piilottaa näkyvät eli ei tarvitse itse tehdä tarkistuksia, onko elementti piilossa vai ei. (jQuery 2015.)

```
function deletebutn (taso){
       //deletes unnecessary buttons in case the user goes back up in the tree
        $(":input[type=submit]").each(function(){

          for(i=0;i<taso.length;i++){

        if($(this).val()==taso[i]){
        $(this).slideUp( "slow" );
       }
     }

  });

}
```

Kuva 7. Animaation lisäys slideUp komennolla

Kuvassa 7 lisätään liu'utus komento ja kuten näkyy, komento on melkein samanlainen kuin haihdutus eli toimivan animaation vaihtaminen on helppoa ja voi kokeilla, kumpi miellyttää omaa silmää enemmän.

4.7.3 Perustehoste-komennot

Edellä mainitut toiminnot eli elementtien piilottamisen, näyttämisen ja vaihtelun piilottamisen ja näyttämisen välillä voi saada aikaan myös ilman animaatioita. Koodissa komennot toimivat täysin samalla lailla kuin animoidut versiot, mutta jos komennolle antaa kestoajan, se muuttuu animoiduksi ja halutessaan tämän animaation saa säätää haluamakseen. Taulukossa 9 esitellään nämä komennot. (jQuery 2015.)

Taulukko 9. Perus tehoste- komennot

Komennon nimi	Komento	Selitys
Piilotus	.hide()	Piilottaa valitun elementin
Näytä	.show()	Näyttää valitun elementin
Kytkin	.toggle()	Piilottaa valitut näkyvät elementit tai näyttää valitut piilotetut elementit

Vaikka puhutaankin perusversioista, saa näihinkin animaatioita kertomalla komennolle suorituksen keston ja halutun vaihdanta-animaation. jQuery UI:n avulla nämä perusversiot saavat paljonkin uudenlaisia mm. animaatioita käyttöönsä. Pelkässä jQueryssa näitä voi pitää animaatiottomina versioina enkä näe syytä muuttaa näitä animoiduiksi, ellei työskentele edellä mainitun jQuery UI-lisäosan kanssa, josta puhutaan myöhemmin (luku 4.10.1).

4.8 Hiiritapahtumat

jQuerya käytettäessä on hyvä tietää, miten hiiren käyttöön liitetään tapahtumia. jQuery tarjoaa hiirenpainalluksille monenlaisia funktioita, jotka on helppo ottaa käyttöön. Useimmat toiminnot ovat samoja, mitä JavaScript tarjoaa, mutta jQuerylla ne uppoavat kivuttomasti yksinkertaisiksi koodipätkiksi.

Taulukossa 10 luodaan yleiskatsaus jQueryn tarjoamiin hiirenkäyttömahdollisuuksiin.

Taulukko 10. Hiiritapahtumat

.click	Hiirennapin klikkaus
.dblclick	Hiirennapin tuplaklikkaus
.mouseover	Hiiren kursori osuu elementtiin
.mouseout	Hiiren kursori poistuu elementistä
.mouseenter	Hiiren kursorin vieminen elementin päälle
.mouseleave	Hiiren kursori viedään pois elementin päältä
.hover	mouseenter ja mouseleave yhdistettynä
.mousedown	Hiiren nappi painetaan alas elementin päällä (mikä tahansa nappi)
.mouseup	Hiiren nappi vapautetaan elementin päällä (mikä tahansa nappi)

Hiiritapahtumia käytetään seuraavasti: ensin valitaan elementti, jonka kanssa voi olla vuorovaikutuksessa ja sen jälkeen liitetään haluttu tapahtuma, jonka laukaisee haluttu hiirenkäyttötapahtuma. (jQuery 2015.) Esimerkkinä kuvassa 8 katsotaan div-elementin häivyttämistä klikkaamalla.

```
$( "div[name=oma]"  ).click(function(){
$(this).fadeOut( "slow" );
});
```

Kuva 8. Click-tapahtuman käyttö

Kuvassa 8 häivytetään valittu div-elementti klikkaamalla sitä. Hiiren klikkaaminen käynnistää funktiokutsun, joka laukaisee häivytys-animaation tälle elementille. Hiiritapahtumien avulla mikä tahansa elementti toimii helposti nappulana.

Seuraavaksi katsotaan omaan työhön lisättyä esimerkkiä mouseenter- ja mouseleave-tapahtumien käytöstä. Kuvan 9 esimerkkikoodi aiheuttaa sen, että aina kun hiiren vie napin päälle, napin tausta muuttuu vihreäksi ja teksti paksunnetaan ja kun hiiren vie pois, nappi palautuu entiselleen.

```
$(":input[type=submit]").mouseenter(function(){
    $(this).css({
      "background-color": "green",
      "font-weight": "bolder" });
    }).mouseleave (function(){
    $(this).css({
      "background-color": "",
      "font-weight": ""});
});
```

Kuva 9. Mouseenter- ja mouseleave-esimerkki

Toiminnon voi myös tehdä hover-tapahtumalla ja se näyttäisi vähän erilaiselta. Hover-tapahtumassa ei tarvitse ketjuttaa mouseenter- ja mouseleave-toimintoja, vaan hover-tapahtuma on näiden yhdistelmä. Esimerkkinä tästä on kuva 10.

```
$(":input[type=submit]").hover(function(){
    $(this).css({
      "background-color": "green",
      "font-weight": "bolder" });
    }, function(){
    $(this).css({
      "background-color": "",
      "font-weight": ""});
});
```

Kuva 10. Hover-tapahtuman esimerkki

Hiiritapahtumissa on hyvä ottaa huomioon, että niiden toiminta saattaa olla erilaista mobiililaitteissa, esimerkiksi mobiililaitteessa hover-funktio näkyy klikatessa jotain elementtiä, missä on hover-toiminto liitettynä mutta vain hyvin pienen ajan ja jos kyseessä on linkki toiselle sivulle, käyttäjä ei välttämättä ehdi edes huomata efektiä ennen kuin sivu on jo vaihtunut. (Lessink 2012.)

Click ja dblclick ovat yksinkertaisia eli hiiren klikkaus tai tuplaklikkaus valitusta elementistä käynnistää määritellyn tapahtuman.

Mouseover-tapahtuma mainittiin aiemmin CSS-ominaisuuksien yhteydessä. Se tapahtuu silloin kun hiiren kursori viedään elementin päälle esimerkiksi, tekstin tyyli muuttuu. Viedessä kursorin pois elementin päältä, se pysyy muuttuneena eikä muutu itsestään takaisin entiseen tyyliin. Tosin, jos haluaa elementin ollessa muuttunut vain silloin, kun hiirenkursori on kohteen päällä, efektille kannattaa käyttää mouseenter ja mouseleave-tapahtumaparia. Tosin kannattaa edelleen muistaa, että mobiilikäyttäjät eivät näistä ominaisuuksista pääse nauttimaan.

Mouseover ja mouseout -paria ei saa sekoittaa mouseenter ja mouseleave -pariin, sillä ne toimivat eri tavalla. Mouseover ja mouseout -komennot koskevat myös valitun elementin sisällä olevia elementtejä eli div-elementin sisällä oleva p-elementti laukaisee omia mouseout ja mouseover -tapahtumia hiiren osuessa tekstiin vaikka se on jo div-elementin sisällä.

Mousedown ja mouseup -tapahtumat ottavat huomioon minkä tahansa hiirennapin painalluksen, myös rullan, minkä kanssa pitää olla tarkkana. Suositeltavaa olisi käyttää yksinkertaista click-toimintoa. (jQuery 2015.)

4.9 Elementtien lisääminen

jQueryn avulla voi myös lisätä elementtejä dokumenttiin. jQueryn tarjoamat mahdollisuudet ovat ulkopuolelle (outside), sisäpuolelle (inside) ja ympärille (around).

Kaikissa alla mainituissa komennoissa, joissa elementtejä luodaan, uuden elementin luontiin on monta tapaa. jQueryn virallisissa dokumenteissa (2015) on käytetty kaikkia seuraavia tapoja.

- $(valitsin).komento (” ”)
- $(valitsin).komento (” ”)
- $(valitsin).komento (" ")
- $(valitsin).komento (” ”)
- $(valitsin).komento (document.createElement(”div”))

Alimpana on jQuery-komennon sisällä oleva natiivi JavaScript-komento.

Alla esitellyt komennot voivat suorittaa koodia, joten siihen liittyy aina tietoturvariski ja jQueryn sivut varoittavat, että näillä metodeilla ei saa suorittaa tekstipätkiä vierailta sivuilta, koska se voi avata cross-site scripting- eli XSS-uhkia. (jQuery 2015.)

4.9.1 Ulkopuolelle

Ulkopuolelle lisääminen tarkoittaa uusien elementtien lisäämistä ennen jo olemassa olevaa elementtiä tai sellaisen jälkeen. On huomioitava, että jo olemassa oleva elementti vaihtaa vain paikkaa eikä lisää uutta elementtiä.

Jälkeen-komento lisää valitun elementin jälkeen määritellyn elementin. Lisättävä elementti annetaan komennon argumenttina esim. $(elementti).after(” Hei ”) lisää elementin jälkeen p-elementin, jonka sisällä on teksti ”Hei”. Sijoita jälkeen-komento toimii samalla tavalla, mutta elementit ja valitsin ovat toisinpäin. Lisättävä elementti tulee valitsimen paikalle ja elementti, jonka jälkeen lisätään, on komennon argumentti. Eli yllä oleva esimerkki menee näin $(” Hei ”).insertAfter(elementti). Syntaksi on ainoa eroavuus komentojen väillä. (jQuery 2015.)

Ennen-komento lisää valitun elementin eteen määritellyn elementin, joka annetaan komennon argumenttina. Tämä toimii samalla tavalla kuin jäl-

keen-komento. Sijoita ennen-komento taas toimii kuten sijoita jälkeen-komento eli lisättävä elementti vaihtaa paikkaa valitsimen kanssa. (jQuery 2015.)

Komennot on koottu taulukkoon 11.

Taulukko 11. Ulkopuolelle lisääminen

Nimi	Komento	Selitys
Jälkeen	.after()	Lisää määritellyn elementin valitun elementin perään
Ennen	.before()	Lisää määritellyn elementin valitun elementin eteen
Sijoita jälkeen	.insertAfter()	Lisää määritellyn elementin valitun elementin perään
Sijoita ennen	.insertBefore()	Lisää määritellyn elementin valitun elementin eteen

4.9.2 Sisäpuolelle

Sisäpuolelle lisääminen lisää elementtien sisään sisältöä, kuten tekstiä. Listassa on komentoja, jotka tekevät samoja asioita mutta eroja on syntaksissa. Uuden elementin lisääminen on mahdollista, kun kirjoittaa argumentiksi oikeaoppista HTML-koodia. Mahdollista olisi myös tallentaa luotava elementti muuttujaan ja käyttää muuttujan nimeä ja lisätä se haluamaansa paikkaan. Sillä, haluaako elementin luoda JavaScriptin createElement(”div”) komennolla vai jollain jQueryn vastaavalla ei ole niinkään väliä, mutta natiivi JavaScript on todennäköisesti hieman suorituskykyisempi.

Komennot HTML ja Teksti toimivat siten, että jos niille ei anna argumentteja, ne palauttavat määritellyn elementin sisällön joko HTML- tai tekstimuodossa. Argumentin annettaessa ne korvaavat määritellyn elementin sisällön annetulla arvolla. (jQuery 2015.)

Lisää loppuun tarkoittaa, että luotava elementti siirtyy määritellyn elementin viimeiseksi lapseksi. Lisää loppuun -komennot append() ja appendTo() ovat toistensa peilikuvia, joissa append-komennossa lisättävä sisältö on komennon argumentti ja taas appendTo-komennossa argumentiksi tulee elementti, jonka loppuun sisältö lisätään. $(elementti).append(” Hei ”) lisää elementin sisään viimeiseksi lapseksi elementin p jonka sisällä on teksti ”Hei”. Sama tapahtuu appendTo-komennolla seuraavasti: $(” Hei ”) appendTo(elementti). (jQuery 2015.)

Lisää alkuun -komennot toimivat samalla tavalla, mutta uudet elementit siirtyvät valitun elementin ensimmäisiksi lapsiksi. Prepend-komento toimii kuten append-komento ja prependTo-komento taas kuten appendTo-komento. (jQuery 2015.)

Komennot on koottu taulukkoon 12.

Taulukko 12. Sisäpuolelle lisääminen

Nimi	Komento	Selitys
Lisää loppuun	.append()	Lisää määritellyn sisällön valitun elementin loppuun
Lisää loppuun	.appendTo()	Lisää määritellyn sisällön valitun elementin loppuun
HTML	.html()	Ottaa elementin HTML sisällön tai korvaa sen määritellyllä sisällöllä
Lisää alkuun	.prepend()	Lisää määritellyn sisällön valitun elementin alkuun
Lisää alkuun	.prependTo()	Lisää määritellyn sisällön valitun elementin alkuun
Teksti	.text()	Ottaa elementin sisällön tekstinä tai korvaa tekstisisällön määritellyllä tekstillä.

4.9.3 Ympäri

Ympärille lisäämisellä tarkoitetaan uuden elementin luomista ja laittamista elementin sisälle ja muutenkin manipuloidaan elementtien järjestystä.

Avaa-komento on yksinkertainen eikä ota edes argumentteja sisäänsä vaan poistaa valitun elementin isäsolmun ja jättää tämän lapset jäljelle. Avaa-komento poistaa vain yhden isäsolmun, ei kaikkia kerralla. Elementille voi siis jäädä isäsolmu, jos se oli tämän isäsolmun lapsenlapsi ennen poistoa. (jQuery 2015.)

Paketoi-komento on avaamisen vastakohta ja luo tai kloonaa jokaiselle valitulle elementille isäsolmun. Isäsolmu määritellään komennon argumenttina ja tämä isäsolmu ilmestyy jokaisen valitsimessa määrätyn elementin ”päälle” ja alkuperäinen elementti sisältöineen on sitten tämän isäsolmun lapsisolmu. Elementin kloonaaminen tapahtuu sillä, että argumenttina käyttää jonkin jo olemassa olevan elementin arvoa eli vaikka luokkaa. Tällöin komento olisi .wrap(” .ElementilleAnnettuLuokka ”), eli argumenttina käytetään valitsinta joka palauttaa halutun, jo olemassa olevan elementin. Kloonattu elementti ei siirry mihinkään vaan siitä ilmestyy klooni. Argumentin arvoksi käyvät myös elementtirakenteet, joissa sisin elementti esiintyy vain kerran. (jQuery 2015.)

Paketoi kaikki -komento luo uuden tai kloonaa olemassa olevan elementin ja hakee kaikki valitsimessa haetut elementit ja lisää ne tämän uuden tai kloonatun elementin alle. Elementti ilmestyy ensimmäisen paketoitavan elementin isäsolmuksi. Tässä pitää olla tarkkana, että valitsin ottaa vain halutut elementit, sillä elementit haetaan koko DOM:sta ja ei haittaa vaikka elementti olisi jo jonkin muun elementin sisällä, se siirretään kylmästi uuden elementin alle. Valitsimena ei siis kannata käyttää elementtien tageja, vaan siirrettävät elementit olisi syytä yksilöidä jotenkin. Kloonaaminen tapahtuu samalla tavalla kuin paketoi- komennon kanssa ja argumentiksi käyvät myös elementtirakenteet, joissa sisin elementti ilmestyy vain kerran. (jQuery 2015.)

Paketoi sisältö -komento luo uuden elementin ja asettaa tämän uuden elementin valitsimessa esiteltyjen elementtien sisällön edelle. Usein tämä tarkoittaa käytännössä sitä, että uusi elementti ilmestyy isän ja lapsen väliin ja lapsi on tässä tilanteessa sisältö eli vaikkapa tekstisolmu. Argumentiksi voi laittaa myös elementtirakenteita mutta sisimmäisen elementin pitää olla rakenteessa vain kerran. .wrapinner( ) lisäisi siis sisällön ympärille p- ja b-tagit eli sisältö olisi nyt p-tagin sisällä lihavoituna. (jQuery 2015.)

Komentojen argumenttien pitää olla oikeaoppista HTML:ää, jossa elementit aukaistaan ja suljetaan oikeassa järjestyksessä. (jQuery 2015.)

Komennot on koottu taulukkoon 13.

Taulukko 13. Ympärille lisääminen

Nimi	Komento	Selitys
Avaa	.unwrap()	Poistaa isäsolmun ja jättää lapsen sen tilalle
Paketoi	.wrap()	Luo jokaisen valitun elementin isäsolmuksi määritellyn elementin
Paketoi kaikki	.wrapAll()	Lisää tietyt elementit määritellyn elementin sisään
Paketoi sisältö	.wrapInner()	Lisää määritellyn elementin tietyn elementin sisällön ympärille

4.9.4 DOM-elementtien poistaminen

DOM.sta voi toki myös poistaa haluamiaan elementtejä. Poistamisessa on pieniä periaatteellisia eroja, joita on hyvä miettiä, kuten tarvitaanko samaa elementtiä myöhemmin.

Irrota-komento poistaa valitun elementin mutta pitää sen tiedot muistissa, esimerkiksi mahdolliset CSS-muutokset, jotta sen voi myöhemmin kutsua takaisin. Komennolle voi antaa argumenttina valitsimen, joka karsii vali-

tuista elementeistä vielä tämän valitsimen perusteella elementtejä. Esimerkiksi $(”div”).detach(”.poista”)-komento poistaa ne div-elementit, joiden luokka on poista. Ilman argumenttia komento poistaisi kaikki div-elementit. (jQuery 2015.)

Tyhjennä-komento tyhjentää valitun elementin lapsisolmut ja poistaa ne eli tilalle jää elementti, jolla ei ole sisältöä.

Poista-komento on täysi poistaminen ja se poistaa sekä valitun elementin, että sen lapsisolmut. Tilalle ei siis jää mitään ja tietty varovaisuus on paikallaan, sillä välillä elementtien suhteiden hahmottaminen on hankalaa. Poista-komennolle voi myös antaa argumentiksi valitsimen, jonka perusteella jo valituista elementeistä valitaan tietty joukko. (jQuery 2015.)

Avaa-komento on sama kuin yllä käyty Ympärille-kappaleen Avaa-komento, eli valitun elementin isäsolmu poistetaan. (jQuery 2015.)

Komennot on koottu taulukkoon 14.

Taulukko 14. DOM:sta poistaminen

Nimi	Komento	Selitys
Irrota	.detach()	Poistaa elementin, mutta pitää sen ominaisuudet muistissa.
Tyhjennä	.empty()	Tyhjentää elementin lapsista, mutta itse elementti jää jäljelle.
Poista	.remove()	Poistaa elementin ja kaiken sen sisällä.
Avaa	.unwrap()	Poistaa isäsolmun ja jättää lapsen sen tilalle

4.9.5 DOM-elementtien korvaaminen

DOM-elementtejä voi myös korvata toisella. Korvattaessa elementtiä ei synny kloonia vaan korvaava elementti siirtyy toisen paikalle ja korvattu elementti poistetaan.

Korvaamiseen on kaksi komentoa, jotka toimivat samalla tavalla ja eroa on vain syntaksin kanssa. ReplaceAll-komennossa korvaava elementti tulee alkuun ja korvattavaksi valitut elementit komennon argumentiksi. ReplaceWith-komento on tavallisempi jQuery-komento, jossa valittu korvattava elementti on alussa ja korvaava elementti on komennon argumentti, komentoa voi myös ketjuttaa muiden komentojen kanssa. ReplaceWith-komento palauttaa jQuery-objektin, joka viittaa korvattavaan elementtiin eikä sen korvaajaan. (jQuery 2015.)

Komentojen käyttäminen itsessään ei eroa muuten muista DOM:n muokkauskomennoista, mutta pitää muistaa, että alkuperäinen sisältö ja kaikki siihen liittyvä tieto, myös sen mahdolliset lapsisolmut, katoavat samalla.

Korvaava elementti voi olla jo olemassa oleva tai se määritellään komennon yhteydessä. Korvatessa olemassa olevalla elementillä, korvaava elementti siirtyy korvattavan paikalle. Korvaava elementti ei ole klooni ja täten poistuu alkuperäisestä paikastaan.

$(" ").replaceAll(".vanha") komento korvaa vanha-luokan omaavat elementit, uusi-nimisellä, tyhjällä div-elementillä.

$(".vanha").replaceWith(" ") tekee saman.

Korvattaessa jo olemassa olevalla elementillä pitää muistaa osoittaa jQuery-objektiin eli $(".vanha").replaceWith($(”.uusi")) jossa kaikkien vanha-luokan omaavien elementtien tilalle haetaan uusi-luokan omaava elementti. Osoittaminen on pakollista, sillä muuten elementti korvattaisiin tekstipätkällä, joka annetaan argumenttina. (jQuery 2015.)

Komennot on koottu taulukkoon 15.

Taulukko 15. Elementtien korvaaminen

Nimi	Komento	Selitys
Korvaa	.replaceAll()	Korvaa valitut elementit määritellyllä elementillä
Korvaa	.replaceWith()	Korvaa valitut elementit määritellyllä elementillä

4.9.6 Elementtien kloonaus

Elementtejä voi myös kloonata ja tätä varten on vain yksi komento ja se esitellään taulukossa 16.

Taulukko 16. Elementin kloonaaminen

Nimi	Komento	Selitys
Kloonaus	.clone()	Kloonaa elementin

Kloonattaessa elementistä ilmaantuu klooni, joka kopioi koko elementin ja sen jälkeläiset ja kaiken tiedon, mikä elementtiin liittyy. Poikkeuksena ovat elementit, joihin käyttäjä voi lisätä sisältöä, esimerkiksi tekstilaatikot ja valintaruudut.. Tekstilaatikot ja valintaruudut itsessään siirtyvät, mutta niiden lisätty sisältö ei.

Kloonauksen tuloksena voi sivulta löytyä kaksi elementtiä, joilla on sama id-arvo ja näin ei saisi käydä, koska id:n pitäisi olla joka elementillä uniikki. Kloonattavilla elementeillä ei siis saisi olla annettuna id-arvoa eikä sitä kannata käyttää kloonattavien elementtien yksilöintiin..

Komennossa ei anneta paikkaa mihin elementti siirretään ja se pitää ketjuttaa komentoon jollain muulla komennolla. Elementti, joka kloonataan, annetaan normaalisti ennen komentoa, mutta paikka pitää määrittää jollain muulla lisäämiskomennolla, kuten appendTo(). (jQuery 2015.) Eli esimerkkilause voisi olla tällainen: $(".oma").clone().appendTo(".vanha"). Oma luokan elementistä tehdään klooni ja klooni lisätään vanha luokan viimeiseksi lapseksi.

4.10 jQueryn lisäosat

jQueryn ympärille on kasvanut muutamia lisäosia, jotka paikkaavat jQueryn puutteita. Tässä käydään läpi kaksi isointa eli käyttöliittymiin keskittyvä jQuery UI ja mobiilisivujen kehittämiseen keskittyvä jQuery Mobile. jQuery UI on lähempänä perinteistä lisäosaa eli lisää jo olemassa oleviin ominaisuuksiin lisää valinnanvaraa, kun taas jQuery Mobile keskittyy lähinnä lisäämään täysin uusia ominaisuuksia.

4.10.1 jQuery UI

jQuery UI on jQueryn lisäosa (plugin) joka vaatii myös jQueryn toimiakseen. jQuery UI lisää käyttöliittymään liittyviä toimintoja ja tehosteita, joita ei löydy pelkästä jQuerystä. jQuery UI lisää myös joihinkin jQueryn toimintoihin lisää mahdollisuuksia, esim. animaatioita CSS-luokkavaihdosten yhteydessä. Osa jQuery UI:n toiminnoista kutsutaan samalla tavalla kuin jQueryn vastaavia, mutta usein niihin saa upotettua animaatioita tai vastaavia lisäominaisuuksia.

jQuery UI tuo myös täysin uusia toimintoja helposti käytettäväksi ja uusia vuorovaikutusmahdollisuuksia tarjotaan viisi kappaletta, jotka ovat:

- Vedettävä elementti (käyttäjä voi muutella elementin paikkaa vetämällä sitä ympäriinsä)
- Pudotettava elementti (tunnistaa, jos päälle pudotetaan vedettävä elementti)
- Kokoa muuttava elementti (käyttäjä voi vetää hiirellä reunasta ja koko muuttuu)
- Valitse listasta (korostaa valitut elementit ja hiirellä voi vetää ja valita joukon listan elementeistä)
- Järjestettävä lista (käyttäjä voi järjestää listaa mielensä mukaan)

Listasta näkee, että uudet elementit ovat aika erikoisia ja se lienee syynä siihen miksi ne on kasattu omaan lisäosaansa. Toki jQuery UI tarjoaa myös pienoisohjelmia (widgets), jotka tarjoavat esim. päivämääränvalitsimen eli kenttää klikattaessa avautuu kalenteri, josta päivämäärä valitaan, sivun sisällä olevia välilehtiä, automaattisesti täyttyviä tekstilaatikoita, joiden sanasto on itse päätettävissä, jne. Valittavissa olevat pienoisohjelmat ovat pieniä mutta käteviä pikku ominaisuuksia.

jQuery UI:n tarjoamissa tehosteissa(effects) päästään asiaa, joka koskettaa itse jQuerya, sillä nämä tehosteet toimivat samalla tavalla, mutta niihin voi

lisätä kätevästi animaatioita. Näitä tehosteita voi lisätä elementtien piilottamisen, näyttämisen ja luokkien poistamisen tai lisäämisen yhteyteen. Näissä on hyvä muistaa, että itse komento on sama kuin jQueryssa mutta UI:n käyttämissä komennoissa on enemmän argumentteja, jotka erottavat komennon jQueryn peruskomennoista. (jQuery 2015.)

On vielä mainittava itse jQuery UI:n lataamisesta. Tiedostoa ladattaessa saa valita, mitä lisäominaisuuksia haluaa mukaan ohjelman kanssa. Vaihtoehdot ovat kategorioittain ja kaikki ominaisuudet ovat valittavissa. Karsiminen tosin kannattaa, sillä jQuery UI:n koko kaikilla herkuilla minimoituna versiona on 235 kilotavua, mikä on noin 2,5 kertaa minimoidun jQueryn koko. Molempien pitää vielä olla liitettynä sivuun, joten jo pelkistä jQuery tiedostoista tulee paljon ladattavaa. Karsiminen siis kannattaa ja ominaisuudet olisi syytä valita niin, että vain tarpeelliset lisäominaisuudet tulevat mukaan.

jQuery UI:n mukana tulee myös valittavissa oleva CSS-teema, kasa ikoneita ja demosivu, joka esittelee ladatut ominaisuudet ja sen, miltä ne näyttävät valitulla teemalla. Halutunlaisen teeman voi myös rakentaa jQueryn omalla ThemeRoller-sivulla, jossa näkee heti, miten elementtien ulkonäkö muuttuu halutun teeman mukaan. ThemeRollerilla voi myös säätää melko vapaasti teemansa CSS-ominaisuuksia tekstin väristä klikattavissa olevien elementtien tekstuureihin. ThemeRoller tarjoaa myös CSS3:n mukaista kulmien pyöristystä, mitä ei tueta Internet Explorer 7 tai 8 -versioissa. (jQuery 2015.)

4.10.2 jQuery Mobile

jQuery Mobile on jQueryn lisäosa, joka keskittyy mobiilisivujen ja sovellusten tekemiseen. jQuery Mobilen tarkoituksena on olla täysin yhteensopiva kaikkien laitteiden ja kaikkien selainten ja käyttöjärjestelmien kanssa. jQuery Mobile vaatii jQueryn toimiakseen, aivan kuten jQuery UI. jQuery Mobile sisältää osin samoja asioita kuin jQuery UI ja voisi sanoa, että jQuery Mobile on jQuery UI mobiilialustalle. jQuery Mobile täydentää jQueryn ominaisuuksia mobiilialustalle sopivaksi ja lisää puuttuvia asioita, kuten tuen pyyhkäisyeleille.

jQuery Mobile sisältää pääosin samat pienoisohjelmat kuin jQuery UI, joitain eroja tosin on, mutta pohjimmiltaan pienoisohjelmat ovat hyvin samanlaisia. jQuery Mobile sisältää toki myös mobiilimaailmasta tuttuja pienoisohjelmia, kuten navigointipalkin.

jQuery Mobile lisää tuen peruseleille, kuten pyyhkäisyt vasemmalle ja oikealle, myös pelkkä pyyhkäisy löytyy. Näiden ominaisuuksia voi myös muuttaa eli voi säätää kuinka monta pikseliä mihinkin suuntaan minkä ajan sisällä lasketaan pyyhkäisyksi. Perusasetus on enemmän kuin 30 pikseliä vaakasuuntaan ja vähemmän kuin 30 pikseliä pystysuuntaan sekunnin sisällä.

Tuet tulevat myös perusvuorovaikutusmahdollisuuksille, kuten kosketus ja yhtäjaksoinen kosketus. Aikaa, joka määrittää yhtäjaksoisen kosketuksen, voi säätää. Perusasetus on 750 millisekuntia.

jQuery Mobile lisää myös tapahtuman, jonka avulla tunnistaa, onko laite pysty- vai vaakasuorassa.

jQuery Mobile tarjoaa samanlaisen download builderin kuin jQuery UI. Kirjoitushetkellä se on vielä alpha-vaiheessa ja se on aika työläs käyttää, sillä valittavaa on paljon ja kuvaustekstit eivät ole parhaasta päästä. Pitää todellakin tietää, mitä tarvitsee ja mitä ei. Tiedoston koko kaikilla ominaisuuksilla minimoitunakin on jopa 197 kilotavua. Download builderista ladattuna mukana ei tule kuvakkeita vaan ne pitää ladata erikseen. Teeman voi tehdä myös ThemeRollerilla ja toiminta on hyvin samanlaista kuin jQuery UI:n vastaava. (jQuery 2015.)

Näistä kahdesta lisäosasta jQuery UI antoi paljon valmiimman ja hiotumman kuvan, kun taas jQuery Mobile vaikutti olevan vielä keskeneräinen, kun download builderkin on vielä alpha-vaiheessa ja dokumentointi ei yllä samalle tasolle jQuery UI:n kanssa.

Myös jQuery Mobilen syntaksi on hieman erilaista kuin taas jQuery UI käyttää täysin samanlaista syntaksia jQueryn kanssa. jQuery Mobilen käyttämä syntaksi muistuttaa jollain tavalla enemmän Android Javaa tapahtumien käsittelijöineen (eventhandler) kaikkineen. Tämä tosin taitaa olla tarkoituksen mukaista, koska lisäosa on tarkoitettu mobiililaitteille ja mobiilikielimäiset mielleyhtymät ovat varmasti asiaan vihkiytyneille tervetulleita.

5 JQUERYN HYÖDYT JA HAITAT

jQueryn hyödyllisyys riippuu paljon siitä, mitä on tekemässä. jQueryn suuri hyöty on, että pienellä koodipätkillä saa aikaan isoja muutoksia. Esimerkkinä vaikka koko sivun taustavärin muuttaminen vie yhden rivin. Koodin vähyyden vuoksi testaaminen ja korjaaminen ovat helpompaa. jQueryn metodit on myös suunniteltu yhteensopiviksi kaikkien selainten kesken. Eli metodit toimivat samalla tavalla selaimesta toiseen, mikä poistaa paljon päänvaivaa kehittämiseltä ja testaamiselta. Kuinka paljon jQuerystä sitten on hyötyä sivuston tai ohjelman kehittämisessä, riippuu tietenkin kehityskohteen koosta. Mitä suurempi projekti, sitä enemmän jQueryn avulla pystyy säästämään koodirivejä, mikä säästää kehitysaikaa.

jQueryn haitoista puhuttaessa, esiin nousee aina suorituskyky, jossa ei päästä puhtaan JavaScriptin tasolle. Mitään suoraa prosenttimäärää, jolla JavaScript päihittää jQueryn ei löydy, sillä nopeusero riippuu käytettävistä metodeista ja hakutavoista. Kuvassa 11 viitataan testiin, jonka on suorittanut Neha Tayal (2014) blogimerkinnässään ”JavaScript vs jQuery: A Quick Overview and Comparison”.

Testing in Chrome 32.0.1700.107 32-bit on Windows Server 2008 R2 / 7 64-bit

Test		Ops/sec
jQuery ID Selector	`var $el = $('#hello');`	1,813,016 ±1.04% 91% slower
JavaScript ID Selector	`var $el = document.querySelector('#hello');`	10,126,325 ±0.40% 48% slower
jQuery Class Selector	`var $el = $('.bye');`	527,960 ±3.48% 97% slower
JavaScript Class Selector	`var $el = document.querySelector('.bye');`	1,623,869 ±0.29% 92% slower
GetElementById	`var $el = document.getElementById('hello')`	19,624,531 ±0.31% fastest

Kuva 11. Suorituskyky vertailu

Hän vertaili JavaScriptin ja jQueryn eroja erilaisissa hakutoiminnoissa. JavaScript on joka haussa nopeampi kuin jQueryn vastaava. Kuinka vahvasti tällaiset erot realisoituvat kehitystyön aikana, on vaikea arvioida ja onko niillä merkitystä tämän päivän tietokoneilla. Mobiilipuolella asia on tietenkin toinen ja hyvä suorituskyky on tärkeää mobiililaitteilla.

jQueryn suorituskyvyn optimointiin on kiinnitetty paljonkin huomiota ja ympäri internetiä jaetaan ahkerasti optimointi kikkoja. jQueryn omassa dokumentaatiossa nostettiin esiin esimerkiksi nimettömien funktioiden käytön välttäminen ja muita valitsimiin liittyviä ja suorituskykyä parantavia ohjeita. Hyvästä suorituskyvystä vastaa kuitenkin ohjelmoija itse ja kaikkien hienompien koodipätkien käyttö pitää jättää sikseen, jos suorituskyvyssä ei päästä halutulle tasolle.

Syy, miksi jQueryn käyttöön päädyttiin työstetyssä sovelluksessa, oli se, minkälaista tukea jQuerylle tarjottiin. Haettaessa vastauksia, miten jokin asia kannattaisi tehdä JavaScriptillä, sai aina vastaukseksi, että tee se näin jQuerylla. Muiden kehittäjien tuki, jota jQueryn käyttöön saa, on ylivertaista JavaScriptiin verrattuna. Syitä voi vain arvailla, mutta veikkaisin, että ventovieraan auttaminen on mukavampaa jos ei tarvitse kirjoittaa isoa läjää koodia, joka ei välttämättä toimi kaikissa selaimissa. Myös käyttäjiä vaikuttaisi olevan todella suuri määrä.

Internetistä löytyy myös erinäisiä pohdintoja, onko JavaScriptin oppiminen enää tarpeellista, koska kaiken voi tehdä vähemmällä vaivalla jQuerylla. Itse en tähän usko, koska jos koodataan jQuerylla, se on kuitenkin vain JavaScriptin toinen ilmenemismuoto. On tärkeää ymmärtää, mitä JavaScriptillä voi tehdä ja ymmärtää sen heikkoudet ja vahvuudet. Lisäksi täytyy ymmärtää, minkälaisen kielen päälle jQuery on rakentunut. JavaScriptiä käytetään hyvin rajattuun asiaan eli selaimissa toimiviin sivuihin ja sovelluksiin ja siinä on, tästä syystä, paljon pieniä ja suurempia eroja tavallisiin ohjelmointikieliin verrattuna. Näiden erojen ja omituisuuksien oppiminen on ehdottoman tärkeää eikä jQuerya oikein osaa arvostaa, jos ei tiedä, miksi ja mistä se on tullut ja mihin sen suosio perustuu.

6 YHTEENVETO

Opinnäytetyö perustuu työharjoittelussa tehtyyn projektiin. Asiakasyrityksen alkuperäisestä ohjelmasta luotiin mahdollisimman alustariippumaton sovellus web-tekniikoita käyttäen. Käytetyt tekniikat olivat JavaScript, HTML, CSS ja jQuery.

Opinnäytetyössäni käytiin läpi jQueryn syntymisen taustoja ja perustoiminnot kuten valitsimet ja niihin liitettävät tapahtumat. Valitsimista käytiin läpi, miten niillä etsitään dokumentista elementtejä. Tapahtumista läpi käytiin DOM:n muokkaus, hiiritapahtumat, sekä CSS ominaisuuksien muokkaus ja se, miten ne liitetään valitsimien kanssa toimiviksi jQuery-koodipätkiksi.

Lisäksi opinnäytetyössä tarkasteltiin lyhyesti myös jQueryn lisäosia jQuery UI ja jQuery Mobile. Näistä lisäosista käytiin läpi tärkeimmät eroavaisuudet, mihin niitä tulee käyttää ja mitä uutta ne tuovat jQueryyn.

Lopuksi käytiin läpi jQueryn hyviä ja huonoja puolia ja mitkä ovat jQueryn suurimmat vahvuudet ja mitkä sen heikkouksia. Suurimmaksi vahvuudeksi nousee käyttäjämäärä ja siitä seuraava vertaistuki kun taas suurin ja eniten pohdintaa aiheuttava puute on jQueryn suorituskyky. Ongelmia voi myös seurata, jos unohdetaan kokonaan, että jQuery on vain JavaScript-kirjasto eikä kokonaan oma kielensä.

LÄHTEET

jQuery Foundation. 2014. Beware Anonymous Functions. jQuery Learning Center. Viitattu 12.1.2015 http://learn.jquery.com/code-organization/beware-anonymous-functions/

jQuery Foundation. 2014. History. Viitattu 4.11.2014. https://jquery.org/history/

jQuery Foundation. 2015. .addclass(). jQuery API. Viitattu 4.5.2015 http://api.jquery.com/addClass/

jQuery Foundation. 2015. .after(). jQuery API. Viitattu 25.4.2015 http://api.jquery.com/after/

jQuery Foundation. 2015. .append(). jQuery API. Viitattu 27.4.2015 http://api.jquery.com/append/

jQuery Foundation. 2015. .appendTo(). jQuery API. Viitattu 27.4.2015 http://api.jquery.com/appendTo/

jQuery Foundation. 2015. .before(). jQuery API. Viitattu 25.4.2015 http://api.jquery.com/before/

jQuery Foundation. 2015. .click(). jQuery API. Viitattu 25.3.2015 http://api.jquery.com/click/

jQuery Foundation. 2015. .clone(). jQuery API. Viitattu 3.5.2015 http://api.jquery.com/clone/

jQuery Foundation. 2015. .dblclick(). jQuery API. Viitattu 25.3.2015 http://api.jquery.com/dblclick/

jQuery Foundation. 2015. .detach(). jQuery API. Viitattu 27.4.2015 http://api.jquery.com/detach/

jQuery Foundation. 2015. .empty(). jQuery API. Viitattu 27.4.2015 http://api.jquery.com/empty/

jQuery Foundation. 2015. .fadeIn(). jQuery API. Viitattu 5.5.2015 http://api.jquery.com/fadeIn//

jQuery Foundation. 2015. .fadeOut(). jQuery API. Viitattu 5.5.2015 http://api.jquery.com/fadeOut/

jQuery Foundation. 2015. .fadeTo(). jQuery API. Viitattu 5.5.2015 http://api.jquery.com/fadeTo/

jQuery Foundation. 2015. .fadeToggle(). jQuery API. Viitattu 5.5.2015 http://api.jquery.com/fadeToggle/

jQuery Foundation. 2015. .hasClass(). jQuery API. Viitattu 4.5.2015 http://api.jquery.com/hasClass/

jQuery Foundation. 2015. .hide(). jQuery API. Viitattu 5.5.2015 http://api.jquery.com/hide/

jQuery Foundation. 2015. .hover(). jQuery API. Viitattu 25.3.2015 http://api.jquery.com/hover/

jQuery Foundation. 2015. .html(). jQuery API. Viitattu 27.4.2015 http://api.jquery.com/html/

jQuery Foundation. 2015. .insertAfter(). jQuery API. Viitattu 25.4.2015 http://api.jquery.com/insertAfter/

jQuery Foundation. 2015. .insertBefore(). jQuery API. Viitattu 25.4.2015 http://api.jquery.com/insertBefore/

jQuery Foundation. 2015. .mousedown(). jQuery API. Viitattu 25.3.2015 http://api.jquery.com/mousedown/

jQuery Foundation. 2015. .mouseenter(). jQuery API. Viitattu 25.3.2015 http://api.jquery.com/mouseenter/

jQuery Foundation. 2015. .mouseleave(). jQuery API. Viitattu 25.3.2015 http://api.jquery.com/mouseleave/

jQuery Foundation. 2015. .mouseout(). jQuery API. Viitattu 25.3.2015 http://api.jquery.com/mouseout/

jQuery Foundation. 2015. .mouseover(). jQuery API. Viitattu 25.3.2015 http://api.jquery.com/mouseover/

jQuery Foundation. 2015. .mouseup(). jQuery API. Viitattu 25.3.2015 http://api.jquery.com/mouseup/

jQuery Foundation. 2015. .prepend(). jQuery API. Viitattu 27.4.2015 http://api.jquery.com/prepend/

jQuery Foundation. 2015. .prependTo(). jQuery API. Viitattu 27.4.2015 http://api.jquery.com/prependTo/

jQuery Foundation. 2015. .remove(). jQuery API. Viitattu 27.4.2015 http://api.jquery.com/remove/

jQuery Foundation. 2015. .removeClass(). jQuery API. Viitattu 4.5.2015 http://api.jquery.com/removeClass/

jQuery Foundation. 2015. .replaceAll(). jQuery API. Viitattu 29.4.2015 http://api.jquery.com/replaceAll/

jQuery Foundation. 2015. .replaceWith(). jQuery API. Viitattu 29.4.2015 http://api.jquery.com/replaceWith/

jQuery Foundation. 2015. .show(). jQuery API. Viitattu 5.5.2015 http://api.jquery.com/show/

jQuery Foundation. 2015. .slideDown(). jQuery API. Viitattu 5.5.2015 http://api.jquery.com/slideDown/

jQuery Foundation. 2015. .slideToggle(). jQuery API. Viitattu 5.5.2015 http://api.jquery.com/slideToggle/

jQuery Foundation. 2015. .slideUp(). jQuery API. Viitattu 5.5.2015 http://api.jquery.com/slideUp/

jQuery Foundation. 2015. .text(). jQuery API. Viitattu 27.4.2015 http://api.jquery.com/text/

jQuery Foundation. 2015. .toggle(). jQuery API. Viitattu 5.5.2015 http://api.jquery.com/toggle/

jQuery Foundation. 2015. .toggleClass(). jQuery API. Viitattu 4.5.2015 http://api.jquery.com/toggleClass/

jQuery Foundation. 2015. .unwrap(). jQuery API. Viitattu 24.4.2015 http://api.jquery.com/unwrap/

jQuery Foundation. 2015. .wrap(). jQuery API. Viitattu 24.4.2015 http://api.jquery.com/wrap/

jQuery Foundation. 2015. .wrapAll(). jQuery API. Viitattu 24.4.2015 http://api.jquery.com/wrapAll/

jQuery Foundation. 2015. .wrapInner(). jQuery API. Viitattu 24.4.2015 http://api.jquery.com/wrapInner/

jQuery Foundation. 2015. :contains() Selector. jQuery API. Viitattu 15.3.2015. http://api.jquery.com/contains-selector/

jQuery Foundation. 2015. :empty Selector. jQuery API. Viitattu 15.3.2015. http://api.jquery.com/empty-selector/

jQuery Foundation. 2015. :even Selector. jQuery API. Viitattu 16.3.2015. http://api.jquery.com/even-selector/

jQuery Foundation. 2015. :first Selector. jQuery API. Viitattu 16.3.2015. http://api.jquery.com/first-selector/

jQuery Foundation. 2015. :has() Selector. jQuery API. Viitattu 15.3.2015. http://api.jquery.com/has-selector/

jQuery Foundation. 2015. :header Selector. jQuery API. Viitattu 16.3.2015. http://api.jquery.com/header-selector/

jQuery Foundation. 2015. :hidden selector. jQuery API. Viitattu 16.3.2015. http://api.jquery.com/hidden-selector/

jQuery Foundation. 2015. :last Selector. jQuery API. Viitattu 16.3.2015. http://api.jquery.com/last-selector/

jQuery Foundation. 2015. :not() Selector. jQuery API. Viitattu 16.3.2015. http://api.jquery.com/not-selector/

jQuery Foundation. 2015. :odd Selector. jQuery API. Viitattu 16.3.2015. http://api.jquery.com/odd-selector/

jQuery Foundation. 2015. :parent Selector. jQuery API. Viitattu 15.3.2015. http://api.jquery.com/parent-selector/

jQuery Foundation. 2015. :visible selector. jQuery API. Viitattu 16.3.2015. http://api.jquery.com/visible-selector/

jQuery Foundation. 2015. All Selector (“*”). jQuery API. Viitattu 15.3.2015. http://api.jquery.com/all-selector/

jQuery Foundation. 2015. Attribute Contains Selector [name*=”value”]. jQuery API. Viitattu 13.5.2015. http://api.jquery.com/attribute-contains-selector/

jQuery Foundation. 2015. Attribute Contains Word Selector [name~=”value”]. jQuery API. Viitattu 13.5.2015. http://api.jquery.com/attribute-contains-word-selector/

jQuery Foundation. 2015. Attribute Ends With Selector [name$=”value”]. jQuery API. Viitattu 13.5.2015. http://api.jquery.com/attribute-ends-with-selector/

jQuery Foundation. 2015. Attribute Equals Selector [name=”value”]. jQuery API. Viitattu 13.5.2015. http://api.jquery.com/attribute-equals-selector/

jQuery Foundation. 2015. Attribute Not Equal Selector [name!=”value”]. jQuery API. Viitattu 13.5.2015. http://api.jquery.com/attribute-not-equal-selector/

jQuery Foundation. 2015. Attribute Starts With Selector [name^=”value”]. jQuery API. Viitattu 13.5.2015. http://api.jquery.com/attribute-starts-with-selector/

jQuery Foundation. 2015. Cache Length During Loops. Viitattu 8.5.2015. http://learn.jquery.com/performance/cache-loop-length/

jQuery Foundation. 2015. Category: Attribute Filter. jQuery API. Viitattu 13.5.2015. http://api.jquery.com/category/selectors/attribute-selectors/

jQuery Foundation. 2015. Class Selector (".class"). jQuery API. Viitattu 15.3.2015. http://api.jquery.com/class-selector/

jQuery Foundation. 2015. Don't Act On Absent Elements Viitattu 8.5.2015. http://learn.jquery.com/performance/cache-loop-length/

jQuery Foundation. 2015. Downloading jQuery. Viitattu 13.5.2015. http://jquery.com/download/

jQuery Foundation. 2015. Element Selector ("element"). jQuery API. Viitattu 15.3.2015. http://api.jquery.com/element-selector/

jQuery Foundation. 2015. Has Attribute Selector [name]. jQuery API. Viitattu 13.5.2015. http://api.jquery.com/has-attribute-selector/

jQuery Foundation. 2015. ID Selector ("#id"). jQuery API. Viitattu 2.3.2015. http://api.jquery.com/id-selector/

jQuery Foundation. 2015. jQuery UI Demos. jQuery UI Viitattu 7.4.2015. http://jqueryui.com/demos/

jQuery Foundation. 2015. Multiple Attribute Selector [name="value"][name2="value2"]. jQuery API. Viitattu 13.5.2015. http://api.jquery.com/multiple-attribute-selector/

jQuery Foundation. 2015. Multiple Selector ("selector1, selector2, selectorN"). jQuery API. Viitattu 3.5.2015 http://api.jquery.com/multiple-selector/

jQuery Foundation. 2015. Optimize Selectors. Viitattu 8.5.2015. http://learn.jquery.com/performance/optimize-selectors/

jQuery Foundation. 2015. orientationchange event. jQuery Mobile API Documentation. Viitattu 25.5.2015. http://api.jquerymobile.com/orientationchange/

jQuery Foundation. 2015. ready(). jQuery API. Viitattu 23.5.2015. http://api.jquery.com/ready/

jQuery Foundation. 2015. swipe. jQuery Mobile API Documentation. Viitattu 13.5.2015. http://api.jquerymobile.com/swipe

jQuery Foundation. 2015. taphold. jQuery Mobile API Documentation. Viitattu 13.5.2015. http://api.jquerymobile.com/taphold/

Korpela, J. 2014. HTML5-käsikirja. Jyväskylä. Docendo.

Kuva 11. Tayal, N. 2014. Perfomance-test JavaScript vs jQuery: A Quick Overview and Comparison. Luce & Morker. http://www.lucemorker.com/blog/javascript-vs-jQuery-quick-overview-and-comparison

Leussink, K. 2012. Mouseover (hover) on touch devices using jQuery. Hnldesign. Viitattu 13.5.2015 http://www.hnldesign.nl/work/code/mouseover-hover-on-touch-devices-using-jQuery/

McPeak, J. 2011. Stop Nesting Functions! (But Not All of Them). Tuts+. Viitattu 4.11.2014. http://code.tutsplus.com/tutorials/stop-nesting-functions-but-not-all-of-them--net-

Swedberg K. 2006. Introducing $(document).ready(). Learning jQuery. Viitattu 23.5.2015. http://www.learningjQuery.com/2006/09/introducing-document-ready

W3Techs. 2015. Historical yearly trends in the usage of JavaScript libraries for websites. Viitattu 14.5.2015 http://w3techs.com/technologies/history_overview/javascript_library/all/y